MASTER EN SCIENCES INFORMATIQUES

Progiciel d'aide à l'enseignement de la biologie par simulation

Dubisy, Françoise

Award date:
1987

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix, Namur

Institut d'Informatique

Année académique 1986 - 1987

PROGICIEL D'AIDE A L'ENSEIGNEMENT DE LA BIOLOGIE PAR SIMULATION

Françoise DUBISY

Mémoire présenté en vue de l'obtention du grade de licencié et maître en informatique

Promoteur : C. CHERTON

Au terme de ce mémoire, mes plus vifs remerciements vont tout naturellement à Monsieur C. CHERTON pour ses conseils judicieux, sa patience à toute épreuve et sa disponibilité tout au long de ce travail.

Je désire tout autant remercier Monsieur E. DEPIEREUX grâce à qui ce mémoire a pu exister. Il se révéla un interlocuteur précieux dans le cadre de ce projet.

Ma reconnaissance va également à tous ceux qui, de près ou de loin, m'ont aidée à mener à bien ce travail.

Dans le système que nous proposons, les résultats des simulations sont mémorisés, ce qui permet de les exploiter de diverses manières. Ils peuvent faire l'objet de manipulations avant d'être par exemple visualisés sous forme graphique, ou encore intervenir dans la détermination d'une nouvelle expérience à simuler.

Nous voulons éviter de proposer un ensemble d'outils disparates et incohérents. Nous nous efforcerons de mettre en évidence les concepts essentiels à nos objectifs, de façon à pouvoir construire un ensemble d'outils intégrés constituant un véritable système compatible avec la notion de simulation.

l'analyse détaillée des concepts fondamentaux, ainsi que des outils à prévoir dans le cadre de ce progiciel, constitue le principal objectif du travail que nous proposons.

Le système ne sera pas entièrement implémenté au terme de ce mémoire. C'est pourquoi l'analyse qui fait l'objet des chapitres suivants est destinée à toute personne qui reprendrait cette tâche.

Nous parlerons désormais de constructeur d'outils pour désigner l'utilisateur associé à cette phase.

Nous avons eu le soucis, en mettant au point notre système, de le laisser le plus ouvert possible. Un certain nombre de concepts seront, comme nous l'avons dit, isolés et définis clairement. Ceci facilite l'extensibilité de notre système.

1.3.2 2ème phase: construction du simulateur

La création du simulateur constitue la seconde étape du cycle de vie d'un didacticiel.

Le simulateur est un programme qui simule une suite d'observations constituant une expérience, et ce sur base d'un modèle mathématique. Le simulateur se charge également de stocker les résultats de la simulation.

Le simulateur est, dans le cadre de ce travail, la notion de base de tout didacticiel. Le simulateur peut être considéré comme le noyau du didacticiel; ce dernier se "contentant" de lui fournir les données nécessaires et d'en exploiter les résultats. C'est pourquoi nous dissocierons l'étape de création du simulateur de l'étape de construction du didacticiel.

Nous parlerons de constructeur de simulateur lorsque nous ferons allusion à l'utilisateur associé à cette phase.

Nous prévoyons dans notre système une série d'outils qui ont pour but d'aider le constructeur du simulateur à mettre au point son programme.

Un simulateur ne peut être exploité que s'il est intégré au sein d'un didacticiel. Tout simulateur est donc destiné à être incorporé à un tel programme. Un même simulateur peut cependant être intégré à plus d'un didacticiel. Il est également possible que plusieurs simulateurs soient exploités par le même didacticiel.

1.3.4 4ème phase: exploitation du didacticiel

Il s'agit de la phase finale du cycle de vie d'un didacticiel, c'est-à-dire son exploitation par l'élève que nous appelerons aussi utilisateur final.

Le rôle de l'élève peut être entièrement passif. Mais dans un but didactique, l'intervention de l'élève semble souvent plus efficace. Il existe diverses manières de faire participer l'élève. Il peut intervenir dans la détermination de l'expérience à simuler, ou encore, choisir la forme sous laquelle il visualisera les résultats de la simulation.

Il semble opportun de préciser que le constructeur d'outils, le constructeur de simulateur, le constructeur de didacticiel et l'utilisateur final ne sont pas forcément quatre personnes différentes. Le simulateur peut être écrit par celui qui met au point le didacticiel dans lequel il sera inclu. D'autre part, il n'est pas exclu que le programmeur utilise son propre produit. Le constructeur d'outils peut lui aussi construire un simulateur ou un didacticiel. Il peut aussi exploiter l'un ou l'autre produit fini.

1.4 Choix du langage et de la machine

1.4.1 Choix du langage

Ce progiciel est destiné à des enseignants du secondaire dont la formation principale n'est pas l'informatique. La plupart d'entre eux se sont ouverts à l'informatique par eux-mêmes, ou en suivant une formation plus ou moins poussée dans ce domaine.

Ces personnes commencent souvent par apprendre à utiliser le langage Basic. Par conséquent, bon nombre de programmes à vocation didactique qu'ils mettent au point sont écrits dans ce langage.

Cependant, le langage Basic présente deux inconvénients majeurs. Le premier est son manque de standardisation. Il n'existe pas de standard auquel on pourrait se référer pour assurer la portabilité du progiciel.

Le second inconvénient est que ce langage ne contient pas les notions de module et procédure. Cette particularité revêt une importance considérable dans le cadre de ce projet où la notion de modularisation est primordiale.

Ceci nous pousse à nous orienter vers un langage de plus haut-niveau. C'est ainsi que notre choix s'est porté sur un langage souvent utilisé lors de l'apprentissage de l'informatique, à savoir le Pascal. Ce langage offre les notions de procédures et fonctions nécessaires à l'implémentation de notre projet. Ce langage a, de plus, une certaine audience dans le secondaire, bien qu'encore relativement réduite.

Vu l'importance grandissante de l'informatique dans l'enseignement, il est à espérer que des langages offrant plus de possibilités que le Basic seront abordés par les enseignants. Nous pouvons donc raisonnablement penser que, dans l'avenir, l'utilisation du Pascal se généralisera. Or, il s'écoulera un laps de temps non négligeable entre le moment où les fondements de ce travail seront établis, et le moment où le système pourrait être opérationnel et mis à la disposition des enseignants. Nous espérons noter d'ici là une évolution dans les langages utilisés.

Le choix du langage fait donc ici l'objet d'un pari sur l'avenir de l'informatique dans les écoles.

Notons que certaines particularités du langage choisi poseront des problèmes lors de l'implémentation de notre système. On peut citer notamment l'absence de tableaux à bornes dynamiques ou l'impossibilité de fournir une procédure en tant que paramètre à une autre procédure. Nous maintiendrons malgré tout notre choix car le langage Pascal est un candidat intéressant dans le cadre de notre projet.

Grâce au système que nous proposons, la rédaction d'un simulateur ou d'un didacticiel sera relativement simple par rapport aux performances qu'on peut attendre. Les constructeurs de simulateurs et de didacticiels écriront leurs programmes en Pascal. Ils pourront faire appel à des outils du progiciel qui seront implémentés dans le même langage.

Précisons encore que c'est le turbo-Pascal que nous avons utilisé au cours de ce travail.

1.4.2 Choix de la machine

Nous désirons implémenter notre système sur un micro-ordinateur, qui est la machine la plus souvent présente dans l'enseignement, et la plus accessible financièrement.

L'évolution des technologies en informatique est telle que le prix du matériel, à l'opposé de celui des logiciels, ne cesse de décroître. Par conséquent, les écoles équipées de micro-ordinateurs depuis quelques années se retrouvent avec du matériel largement dépassé (Apple II, TRS 80,...).

Les Apple II et autres machines de même niveau sont encore très utilisées dans les écoles, mais n'offrent pas la capacité mémoire suffisante pour notre projet. D'autre part, un certain nombre d'établissements scolaires se sont équipés d'un matériel plus moderne. Nous préférons, par conséquent, opter pour les P.C. compatibles qui ont déjà fait leur apparition dans certaines écoles, et qui, dans ce type d'environnement, semble devenir une sorte de nouveaux standards. Ce type de micro-ordinateur permet de disposer d'une capacité mémoire suffisante, à l'inverse de beaucoup de machines disponibles actuellement dans le secondaire. Nous avons travaillé sur une classe particulière de machines compatibles à savoir les olivetti M 24.

CHAPITRE 2 : CONCEPTS RELATIFS AU SIMULATEUR

2.1 Phénomène biologique, modèle mathématique et simulation

Un phénomène biologique, faisant partie de l'univers biologique, est un phénomène uniquement observable. En effet, les règles (ou lois) régissant les phénomènes étudiés sont inconnues à priori; on ne peut qu'en observer les effets par un nombre limité d'expériences et par analyse des observations ainsi faites. Un certain nombre de lois ou règles peuvent alors être induites.

A l'inverse, l'univers mathématique est constitué d'un ensemble de lois de base données a priori et où toute autre chose sera tenue pour vraie si elle peut être déduite à partir de ces lois.

Lors de l'analyse de phénomène biologique, les scientifiques ont constaté que des composants de la réalité biologique semblent se comporter de même façon que des êtres de la réalité mathématique. C'est ainsi qu'une correspondance entre les deux univers peut être proposée. Une telle correspondance est appelée "modèle". On dispose alors d'un modèle mathématique de la réalité biologique considérée (voir figure 2.1).

Le principe de base des sciences empirico-formelles est de considérer que les relations ainsi constatées restent valables dans les cas où l'on n'a pas fait l'expérience, élevant ainsi ces relations au rang de lois générales. Les modèles mathématiques peuvent alors être utilisés à des fins de prédiction.

L'observation du modèle mathématique en lieu et place du phénomène biologique porte le nom de simulation. La démarche traditionnelle est de type inductive, à l'opposé de la simulation de phénomène biologique sur base de modèle mathématique qui est de type déductive.

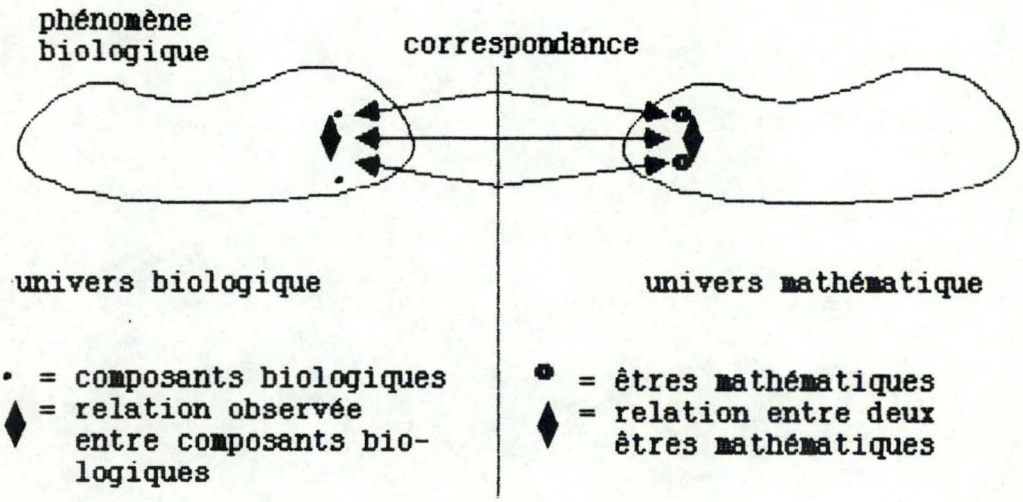


figure 2.1.

Cette dernière démarche est entre autres intéressante dans le domaine de l'enseignement assisté par ordinateur. Plutôt que de travailler directement sur la réalité (prélèvements, observations, expériences,...), celle-ci est simulée à l'aide de modèles mathématiques. Cette démarche est particulièrement intéressante dans les situations où l'expérimentation est coûteuse, malaisée ou dangereuse.

Dans le cadre de ce travail, nous nous limiterons à des modèles numériques. Dans de tels modèles, les observations effectuées sont représentées par des nombres ou groupes de nombres.

Nous proposons ci-dessous une brève description de quelques modèles biologiques existants. Nous les donnons surtout à titre d'exemple. Nous en reprendrons l'un ou l'autre dans la suite pour illustrer certains des concepts que nous introduirons.

Bon nombre de modèles biologiques sont décrits par une ou plusieurs équations différentielles. S'il n'est pas possible de trouver une solution analytique aux équations différentielles du modèle, on peut avoir recours à l'intégration numérique.

On peut dissocier les modèles simples des modèles à composants multiples.

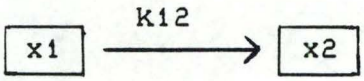
Parmis les modèles simples, certains sont décrits par équation différentielle et peuvent être résolus analytiquement par intégration. Prenons comme exemple le modèle de variation de la densité de la population (N) en fonction du temps (t): $dN/dt = kN$, où k est une constante de proportionalité. La solution analytique de ce modèle est de la forme: $N_t = N_0 . e^{kt}$, si N_0 est la population au temps $t = 0$ ([SPAIN] p. 16).

D'autres modèles (simples), partent de l'hypothèse que le système biologique est à l'équilibre ("steady-state"). Le modèle de saturation enzymatique de Michaelis-Menten en est un exemple. La solution analytique est de la forme:

$$v = \frac{VMax. [S]}{Km + [S]}$$

où km est la constante de Michaelis-menten, v est la vitesse de la réaction, [S] est la concentration en substrat et VMax est la constante représentant la vitesse maximale lorsque l'enzyme est totalement saturé. La vitesse de la réaction enzymatique est donc exprimée en fonction de la concentration en substrat. En ce qui concerne le développement et les calculs, le lecteur peut se référer à [spain], p. 28.

Les modèles à composants multiples sont souvent utilisés en biologie. Prenons un modèle à deux compartiments entre lesquels s'effectue un flux de matière. Ce modèle peut être schématisé de la manière suivante:



Ce modèle est décrit par le système d'équations différentielles:

$$\begin{cases} dx1/dt = -k12 . x1 \\ dx2/dt = +k12 . x1 \end{cases}$$

où x1 et x2 sont des quantités de matière, et k12 une constante de proportionalité liée au flux entre les deux compartiments. La solution analytique de ce système est:

$$\begin{cases} x1 = x1(0) . e^{-k12t} \\ x2 = x2(0) + x1(0) . (1 - e^{-k12t}) \end{cases}$$

où t représente le temps, et x1(0) et x2(0) sont les quantités de matière dans chaque compartiment au temps t = 0 ([lebret], p. 101). Les quantités de matière x1 et x2 sont donc exprimées en fonction du temps.

Lorsqu'il est malaisé de trouver une solution analytique à un système d'équations différentielles, on peut avoir recours à l'intégration numérique. Prenons, par exemple, le modèle (à multi-composants) de la cinétique d'une réaction bimoléculaire de la forme:

$$A + B \xrightarrow{K} P$$

Les équations différentielles sont:
$$\begin{cases} d[A]/dt = -K.[A].[B] \\ d[B]/dt = -K.[A].[B] \\ d[P]/dt = +K.[A].[B] \end{cases}$$

La solution numérique est:
$$\begin{cases} \Delta[B] = \Delta[A] = -K.[A].[B].\Delta t \\ \Delta[P] = +K.[A].[B].\Delta t \\ [A]_{t+\Delta t} = [A]_t + \Delta[A] \\ [B]_{t+\Delta t} = [B]_t + \Delta[B] \\ [P]_{t+\Delta t} = [P]_t + \Delta[P] \end{cases}$$

où [A], [B] et [P] sont les concentrations des différents composants, K est une constante (taux de réaction) de même que Δt (accroissement du temps). Les valeurs de chaque concentration au temps $t+\Delta t$ dépend de la concentration au temps t ([SPAIN], p. 81).

Notons que certains modèles à composants multiples peuvent aussi partir de l'hypothèse que le système est à l'état d'équilibre pour proposer une solution de type analytique.

La description de certains modèles, qu'ils soient simples ou à composants multiples, ne passe pas forcément par l'écriture d'équations différentielles. C'est par exemple le cas du modèle simple de variation de l'intensité lumineuse d'un cycle annuel, décrit par l'équation:

$$I = 249 \sin(2\pi(W-11)/52) + 378,$$

où I est l'intensité de la radiation directe en calories par cm par jour, et W = le temps en semaine par rapport au calendrier ([spain], p. 135).

Les types de modèles que nous venons brièvement de décrire sont déterministes. D'autres modèles font intervenir une dimension aléatoire. C'est notamment le cas lorsqu'on désire simuler la variabilité expérimentale. Ces modèles non

déterministes peuvent se baser sur la génération de nombres aléatoires de distribution donnée pour simuler cette composante aléatoire. Nous en reparlerons au chapitre 5.

Les types de modèles evoqués ci-dessus sont compatibles avec le système que nous proposons dans la suite de ce travail. Ceci ne signifie pas que d'autres types de modèles ne puissent pas être compatibles avec notre système. Nous ferons éventuellement allusion à d'autres types de modèles au cours de ce travail, ce sera notamment le cas lorsque nous parlerons de variable indépendante non connue a priori.

2.2 Calculateur et variables dépendantes, indépendantes et évolutives

Dans l'univers biologique, l'expérimentateur peut observer un même phénomène grâce à divers dispositifs expérimentaux. A un même modèle mathématique peuvent donc être associés plusieurs dispositifs expérimentaux simulés.

Dans un dispositif expérimental, il existe un certain nombre de variables. Nous appelerons variable une composante du phénomène observé qui, dans un dispositif expérimental particulier, peut prendre des valeurs différentes lors d'observations différentes.

Dans le cadre de ce travail, nous définirons un calculateur comme étant un programme, au sens informatique du terme, qui, sur base d'un modèle mathématique, simule une observation d'un phénomène biologique (voir concept de simulation d'observation), et ce, selon un seul dispositif expérimental particulier.

2.2.1 Dispositif expérimental de type statique

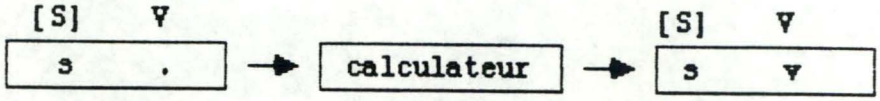
Le premier type de dispositif expérimental comprend d'une part des variables qui sont connues avant l'observation, et d'autre part, des variables mesurables ou calculables. Nous lui donnerons le nom de dispositif expérimental statique. Nous appelerons variable indépendante toute variable dont la valeur doit être connue ou fixée dans un dispositif expérimental avant chaque observation. Nous appelerons variable dépendante toute variable mesurée dans (ou par) le dispositif expérimental. Dans un tel dispositif, si l'on fixe un certain nombre de variables (les variables indépendantes), les variables dépendantes s'en trouvent déterminées.

Un exemple de tel dispositif est donné à la figure 2.2.

Soit le modèle de saturation enzymatique de Michaelis-Menten

$$E + S \underset{k2}{\overset{k1}{\rightleftharpoons}} ES \underset{k4}{\overset{k3}{\rightleftharpoons}} E + P$$

$$\rightarrow V = \frac{VMax \,.\, [S]}{Km + [S]} \quad \text{avec } Km = \frac{k2 + k3}{k1} = \text{constante de Michaelis-Menten}$$



où VMax et Km sont des constantes
[S] = variable indépendante
V = variable dépendante
. = variable inconnue
s = valeur connue
v = valeur calculée

Dispositif expérimental de type statique

figure 2.2.

Le modèle de variation de la densité de la population: $N_t = N_0 \,.\, e^{kt}$ dont il a été question plus haut, rentre également dans cette catègorie. N_0 et k sont des constantes, t est une variable indépendante et N_t est une variable dépendante.

Il en est de même pour le modèle à deux compartiments dont la solution analytique est:

$$\begin{cases} x1 = x1(0) \,.\, e^{-k12t} \\ x2 = x2(0) + x1(0)\ (1 - e^{-k12t}) \end{cases}$$

x1(0), x2(0) et k12 sont des constantes, t est une variable indépendante, et x1 et x2 sont des variables dépendantes.

Nous verrons plus loin comment prendre en compte les constantes (voir calculateur dérivé).

Dans ces types de modèles, on prend en compte le temps, en le considérant comme une variable indépendante. On observe alors une suite d'états du système à différents instants.

Dans un tel type de dispositif expérimental, le calculateur reçoit en entrée un vecteur contenant différentes variables (dépendantes et indépendantes), les valeurs des variables indépendantes étant connues. le calculateur, après exécution, fournit le même vecteur dans lequel les valeurs des variables indépendantes sont intactes, et les valeurs des variables dépendantes sont calculées.

On peut alors considérer que le calculateur calcule un état du système.

2.2.2 Dispositif expérimental de type dynamique

Dans un second type de dispositif expérimental, nous considérons un système effectuant des transitions d'un état vers un autre.

Nous parlerons alors de dispositif expérimental dynamique.

Dans ce type de système, il est fréquent que le modèle utilisé fournisse les valeurs des variables, lors d'une observation, en fonction des valeurs de ces mêmes variables obtenues lors de l'observation précédente. Nous appelerons variables évolutives ce type de variables.

On peut alors considérer qu'un calculateur simule une transition d'état d'un système dynamique en remplaçant les valeurs des variables évolutives par leur valeur mise à jour. On peut illustrer ceci par un exemple (voir figure 2.3).

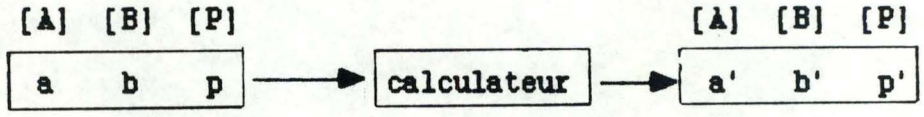
Si l'on désire étendre la notion de dispositif expérimental dynamique, on peut admettre qu'avant chaque observation, un certain nombre de paramètres aient leur valeur fixée a priori. Ces paramètres pourraient par exemple définir l'intervalle de temps entre deux observations consécutives, ou encore, correspondre à des réglages successifs d'un dispositif de contrôle. Ces paramètres peuvent être considérés comme des variables indépendantes.

Supposons la réaction chimique suivante: $A + B \xrightarrow{k} P$
où A et B sont des réactifs
P est un produit
k est le taux de réaction.

Les équations qui en découlent et qui constituent le calculateur sont:

$$\Delta[B] = \Delta[A] = -k.[B].[A].\Delta t$$
$$\Delta[P] = +k.[B].[A].\Delta t$$
$$[B]_{t+\Delta t} = [B]_t + \Delta[B]$$
$$[A]_{t+\Delta t} = [A]_t + \Delta[A]$$
$$[P]_{t+\Delta t} = [P]_t + \Delta[P]$$

où Δt = intervalle de temps



avec k et Δt = constantes
[A],[B],[P] = variables évolutives
a, b, p = valeurs des variables évolutives avant la transition d'état
a',b',p' = valeurs des variables évolutives après la transition d'état

figure 2.3.

Dans un tel dispositif, le vecteur fourni au calculateur contient des variables indépendantes et évolutives, les valeurs de ces variables étant connues. Le calculateur, après exécution, fournit le même vecteur dans lequel les variables évolutives sont mises à jour.

Il n'est pas exclu de retrouver au sein d'un même dispositif expérimental les trois types de variables, à savoir indépendantes, dépendantes et évolutives (voir figure 2.4).

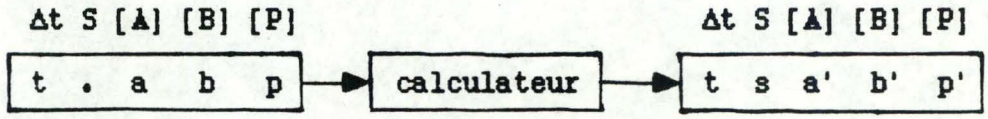
Supposons la réaction chimique suivante: $A + B \xrightarrow{k} P$

où A et B sont des réactifs
P est un produit
k est le taux de réaction.

Les équations qui en découlent et qui constituent le calculateur sont:

$$\Delta[B] = \Delta[A] = -k.[B].[A].\Delta t$$
$$\Delta[P] = +k.[B].[A].\Delta t$$
$$[B]_{t+\Delta t} = [B]_t + \Delta[B]$$
$$[A]_{t+\Delta t} = [A]_t + \Delta[A]$$
$$[P]_{t+\Delta t} = [P]_t + \Delta[P]$$
$$S = [A]_{t+\Delta t} + [B]_{t+\Delta t}$$

où Δt = intervalle de temps



avec k = constante
Δt = variable indépendante
S = variable dépendante
. = valeur inconnue
s = valeur calculée
t = valeur connue
[A],[B],[P] = variables évolutives
a, b, p = valeurs des variables évolutives avant la simulation de l'observation
a',b',p'= valeurs des variables évolutives après la simulation de l'observation

figure 2.4.

Les variables indépendantes telles que nous les avons définies jusqu'ici sont connues a priori, c'est-à-dire avant la simulation de l'expérience (voir concept de simulation d'expérience).

On peut étendre les mécanismes de détermination des variables indépendantes en admetant que:

-la valeur d'une variable indépendante peut être égale à la valeur prise par une autre variable à l'observation précédente

-la valeur d'une variable indépendante peut être déterminée par calcul à partir d'une (éventuellement plusieurs) valeur obtenue à l'observation précédante

-la valeur d'une variable indépendante peut être calculée à partir de valeur(s) obtenue(s) lors d'observation(s) antérieure(s) à la dernière observation.

Puisque les valeurs de ces variables sont connues lorsqu'elles sont fournies au calculateur, nous pouvons les ranger parmi les variables indépendantes. Les valeurs de ces variables font l'objet d'un calcul qui ne peut être effectué qu'avant chaque exécution du calculateur. Elles diffèrent donc par leur mode de détermination des variables indépendantes connues a priori.

Nous pouvons résumer ce qui vient d'être exposé de la manière suivante.

Un calculateur est un programme qui reçoit en entrée un vecteur comprenant les valeurs des variables qui peuvent être de trois types: dépendantes, indépendantes et évolutives. Les valeurs des variables indépendantes et évolutives du vecteur fourni en entrée sont connues, tandis que les valeurs des variables dépendantes ne le sont pas. Le calculateur, après exécution, fournit le même vecteur dans lequel les valeurs des variables indépendantes sont intactes, les valeurs des variables dépendantes sont calculées, et les valeurs des variables évolutives sont mises à jour.

Le calculateur étant un programme, les variables reprises dans le vecteur sont des paramètres au sens informatique du terme.

Rappelons qu'à un modèle mathématique peuvent être associés plusieurs dispositifs expérimentaux. Par conséquent, un même modèle mathématique peut donner naissance à plus d'un calculateur: au moins un calculateur par dispositif expérimental simulé. Une variable indépendante dans un calculateur peut être variable dépendante dans un autre calculateur, tous deux associés au même modèle mathématique mais à des dispositifs expérimentaux différents. En d'autres termes, une variable peut être connue a priori dans un dispositif expérimental, mais être mesurée dans un autre dispositif expérimental associé au même modèle mathématique, et inversément (voir figure 2.5).

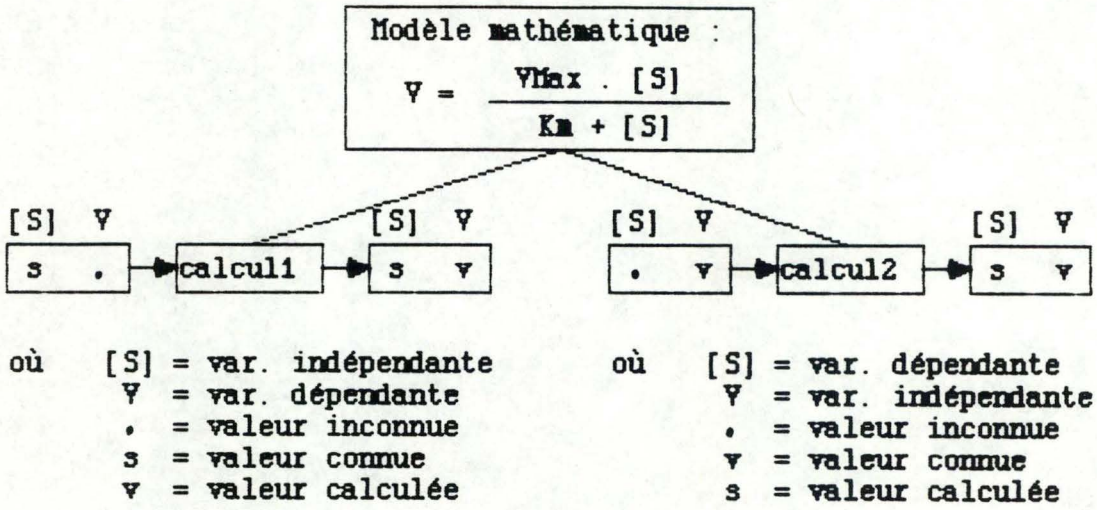


figure 2.5.

Dans le cas de modèles non déterministes, les principes énoncés ci-dessus restent valables. C'est au sein du calculateur que la dimension aléatoire sera prise en compte. Des nombres générés aléatoirement (suivant une distribution donnée) peuvent être utilisés pour moduler les valeurs des variables calculées dans le cas où par exemple on désire simuler la variabilité expérimentale.

2.3 Simulation d'observation et simulation d'expérience

Nous appelerons simulation d'une observation une exécution d'un calculateur, avec affectation préalable de valeurs aux différentes variables indépendantes et évolutives de ce calculateur. On entend par exécution d'un calculateur le calcul des valeurs des variables dépendantes et la mise à jour des variables évolutives à partir des valeurs reçues: valeurs des variables indépendantes et évolutives.

Nous appelerons simulation d'une expérience une série d'exécutions du même calculateur avec, avant chaque exécution, affectation de valeurs aux différentes variables indépendantes et évolutives de ce calculateur. La simulation

d'une expérience est donc une série de simulations d'observation, utilisant le même calculateur (voir figure 2.6).

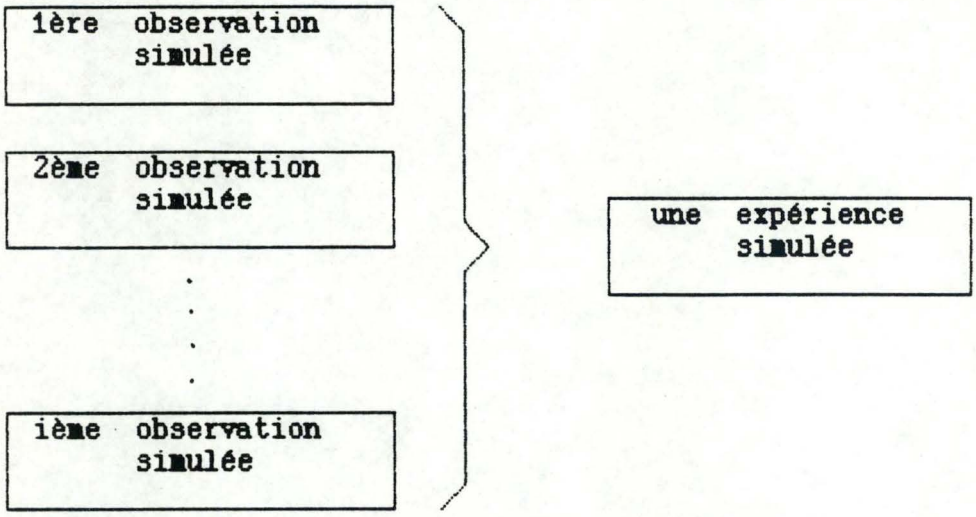


figure 2.6.

2.4 Calculateur dérivé

Un calculateur dérivé est un calculateur construit à partir d'un calculateur préexistant dont certains paramètres (au sens informatique du terme) sont fixés, donc constants tout au long d'une expérience simulée, ou "ignorés".

Remarquons qu'une constante dans un dispositif expérimental peut être considérée comme une variable indépendante dont la valeur est fixe tout au long de la simulation de l'expérience. On peut donc écrire un calculateur dérivé dans les paramètres duquel la variable constante tout au long de l'expérience n'apparait plus (voir figure 2.7).

Un paramètre peut être ignoré et ne pas être repris au niveau du calculateur dérivé si ce paramètre n'est pas jugé utile ou nécessaire. De même, une variable dépendante faisant partie du vecteur de variables fourni à un calculateur donné peut ne pas être reprise dans les paramètres du calculateur dérivé de ce dernier. Ce peut être le cas notamment lorsque les valeurs obtenues pour cette variable au cours de l'expérience simulée ne sont pas

exploitées par la suite, ou n'interviennent pas dans le calcul d'autres variables (revoir figure 2.7).

Soit un calculateur de base :

```
procedure  C (k, t:real; var S, [A], [B], [P]:real);
     begin
        .
        .
        .
     end;
```

où k et t = variables indépendantes
S = variable dépendante
[A],[B],[P] = variables évolutives;

soit un calculateur C' dérivé de C :

```
   procedure  C' (var [A], [B], [P]:real);
      var S:real;
      begin
         C (0.2, 0.1, S, [A], [B], [P]);
      end;
```

où [A] ,[B] ,[P] = variables évolutives
S = ignoré
k et t = constantes respectivement égales 0.2 et 0.1

figure 2.7.

Dans la suite de ce texte, nous ne ferons plus de distinction entre calculateur de base et calculateur dérivé.

2.5 Tableau de résultats

Les résultats obtenus en simulant une expérience doivent pouvoir être visualisés sous forme de graphe, ou simplement sous forme de tableau contenant les valeurs des différentes variables au cours d'une expérience simulée. L'expérimentateur peut utiliser ces résultats pour définir d'autres expériences à simuler. Des statistiques (moyennes, variances,...) sur les résultats de la simulation d'une expérience peuvent également être calculées (voir manipulation de tableaux).

Ces quelques propositions montrent qu'il serait intéressant de garder trace des différentes variables, c'est-à-dire des valeurs prises successivement tout au long d'une même expérience simulée. Ceci introduit la notion de tableau de résultats. Un tableau de résultats est constitué d'une part d'un vecteur que nous appelerons vecteur-définition et, d'autre part, d'un tableau à deux dimensions que nous appelerons matrice.

Le vecteur-définition reprend pour chaque variable du tableau de résultats une série de renseignements: la définition de la variable. Cette définition consiste en :

- un nom
- une unité
- un genre
- un ensemble d'informations

Tout nom de variable est identifiant au sein d'un même tableau, c'est à dire que deux variables d'un même tableau de résultats ne peuvent porter le même nom. L'unité est associée aux valeurs de la variables reprises dans la matrice (cfr infra). Le genre de la variable peut être: "dépendante", "indépendante" ou "évolutive".

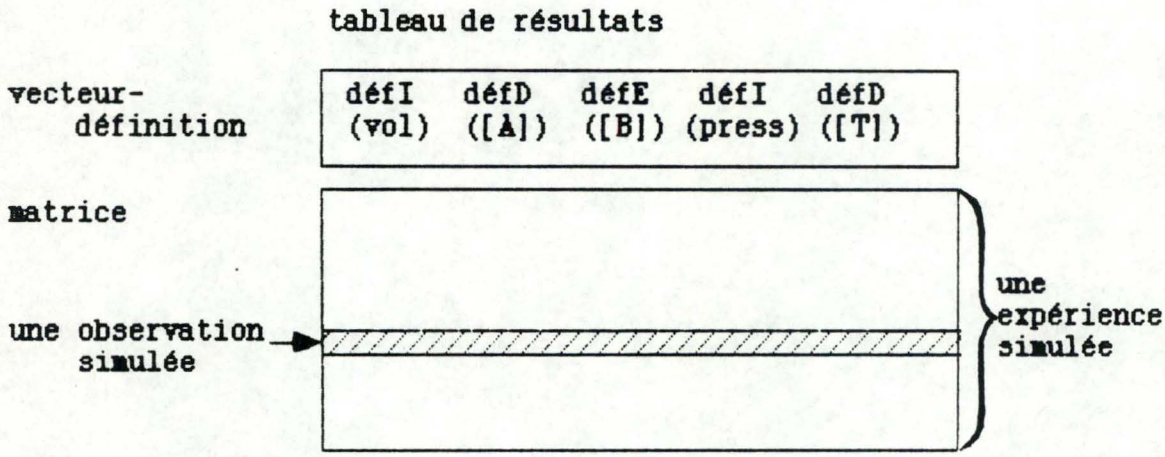
Les informations associées à une variable dépendent du genre de cette dernière. Le genre "dépendante" ne nécessite aucune information complémentaire. L'information associée au genre "évolutive" n'est autre que la valeur à affecter à cette variable avant la première observation à simuler. Cette valeur représente l'état initial de la variable évolutive. Quant aux informations associées au genre "indépendante", elles seront abordées plus loin (voir simulateur).

La matrice d'un tableau de résultats mémorise les valeurs prises par les différentes variables au cours d'une expérience simulée. Chacune des valeurs reprises dans le

vecteur fourni, <u>après</u> chaque exécution, par le calculateur, est mémorisée dans une case de la matrice. Chaque ligne de la matrice contient donc les valeurs prises, <u>après</u> chaque exécution du calculateur (c'est-à-dire à chaque observation simulée), par les variables. Chaque colonne contient les différentes valeurs prises par une même variable, tout au long d'une même expérience simulée.

La matrice renferme autant de colonnes qu'il y a de variables associées au calculateur utilisé, et autant de lignes qu'il y a d'observation(s) simulée(s) dans une expérience simulée.

Le vecteur-définition d'un tableau de résultats reprend autant de définitions de variables qu'il y a de colonnes dans la matrice associée. L'ordre des définitions de ce vecteur et l'ordre des colonnes de la matrice est significatif. En effet, la ième définition dans le vecteur-définition correspond à la variable dont les valeurs sont reprises dans la ième colonne de la matrice associée. A chaque simulation d'expérience est donc associé un tableau de résultats (voir figure 2.8).



défI = définition de variable indépendante
défD = définition de variable dépendante
défE = définition de variable évolutive

Les noms de variables sont mis entre parenthèses

<u>figure 2.8.</u>

2.6 Schéma expérimental

Rappelons que l'on distingue deux modes de détermination des variables indépendantes. La valeur d'une variable indépendante peut soit être connue a priori, c'est-à-dire avant le début de l'expérience, soit être calculée à partir de valeur(s) obtenue(s) lors d'observation(s) antérieure(s) à l'observation courante au sein de la même expérience simulée. Ce second mode de détermination de variable indépendante sera approfondi plus loin (voir concept de simulateur).

Un schéma expérimental est constitué, d'une part, de l'ensemble des valeurs des variables indépendantes connues a priori, et, d'autre part, de la description de l'état initial du système. Cet état doit être décrit lorsqu'au calculateur est associée au moins une variable évolutive ou indépendante non connue a priori. Cet état est déterminé par les valeurs à affecter aux variables évolutives avant de simuler la première observation, ainsi que par les valeurs de base à affecter aux variables indépendantes non connues a priori (voir concept de simulateur).

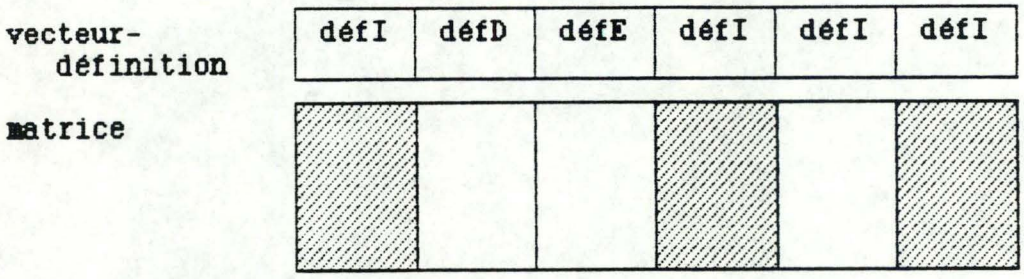
La création d'un schéma expérimental consiste, en ce qui concerne les variables indépendantes connues a priori, à initialiser les colonnes d'une matrice correspondant à ce genre de variable. On entend par "initialiser une case d'une matrice", y inscrire une valeur (voir figure 2.9). Les valeurs d'une variable indépendante connue a priori reprises dans un schéma expérimental seront affectées à cette variable et fournies une à une au calculateur (via le vecteur) avant chacune de ses exécutions.

Il y a évidemment, dans une colonne de la matrice correspondant à une telle variable, autant de valeurs qu'il y a d'observations au sein de l'expérience à simuler.

La façon la plus simple de construire la partie du schéma expérimental correspondant aux variables indépendantes connues a priori est de procéder comme suit. Dans un premier temps, il faut créer différentes matrices à une colonne (ou matrices-colonne), chaque matrice contenant les valeurs à affecter à une même variable. Il suffit ensuite de constituer une nouvelle matrice par juxtaposition des différentes matrices-colonne (voir figure 2.10).

Il est évident que les colonnes juxtaposées en vue de constituer un schéma expérimental doivent avoir le même nombre d'éléments. Des outils de création , ainsi qu'un outil de juxtaposition de matrices paraissent nécessaires.

tableau de résultats

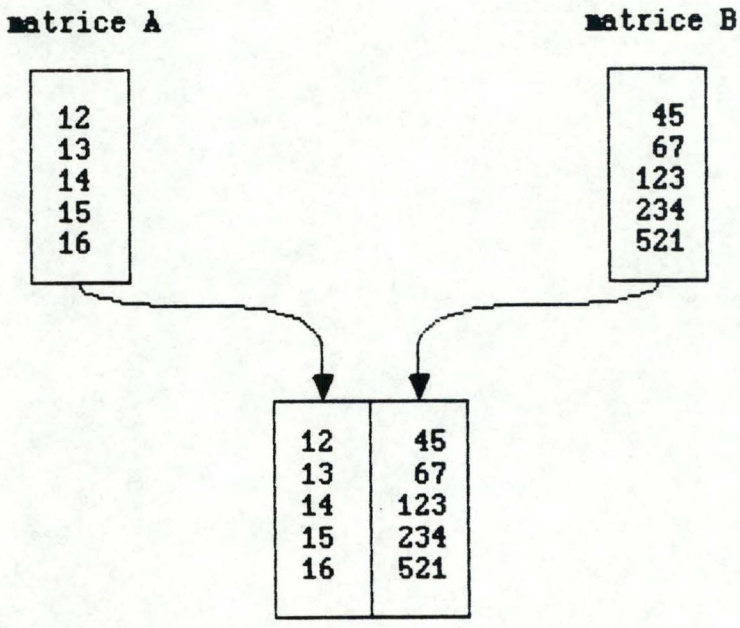


= valeurs de variable indépendante connue a priori

défI = définition de variable indépendante
défD = définition de variable dépendante
défE = définition de variable évolutive

Les trois colonnes hachurées représentent la partie du schéma expérimental correspondant aux variables indépendantes connues a priori.

figure 2.9.



Construction d'une nouvelle matrice par juxtaposition

figure 2.10.

Une autre technique utile dans la construction de schéma expérimental consiste à effectuer le produit cartésien ordonné de deux matrices. Illustrons ceci par un exemple portant sur deux matrices-colonne.

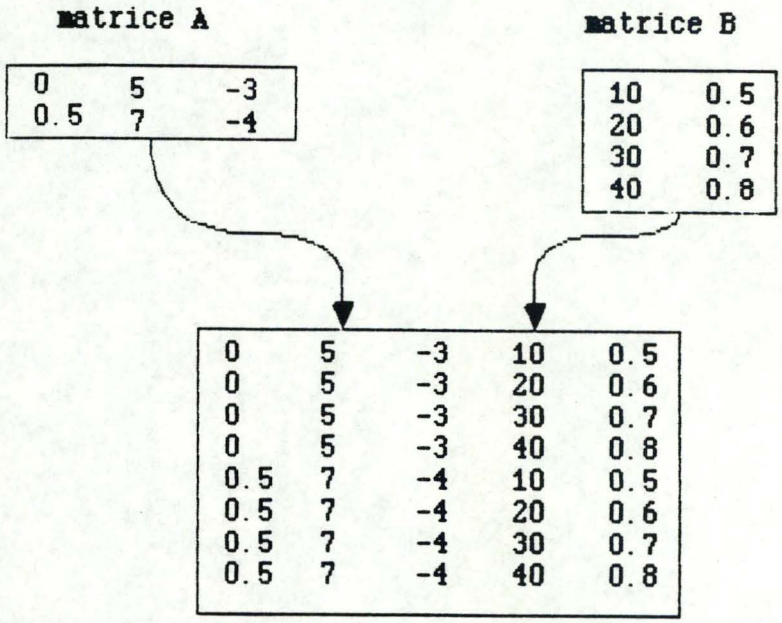
Supposons un schéma expérimental constitué de deux variables indépendantes connues a priori, soient A et B, dont la suite des valeurs ont été définies par l'expérimentateur et contiennent respectivement les valeurs (2, 4, 6) et (0, 1, 2). Supposons, que l'expérimentateur ait défini de telles suites, parce qu'il désire observer le modèle mathématique en lieu et place d'un phénomène biologique soumis à certaines contraintes, exprimées par A = 2, 4 ou 6 et par B = 0, 1 ou 2. Si pour chacune des contraintes imposées par les valeurs affectées à A, l'expérimentateur désire observer le modèle mathématique en réponse à ces contraintes, pour toutes les valeurs de B, alors pour chaque valeur affectée à A, trois observations seront simulées, correspondant respectivement aux trois valeurs de B.

Ce principe peut être généralisé. Une matrice peut être constituée par produit cartésien ordonné de deux matrices à plus d'une colonne (voir figure 2.11). Elle peut aussi être construite en partie par produit cartésien ordonné et en partie par juxtaposition de matrices.

Remarquons qu'il serait également intéressant de pouvoir constituer une matrice-colonne en recopiant intégralement et dans le même ordre les valeurs d'une colonne d'une matrice déjà existante (voir figure 2.12). Un interfaçage entre différents tableaux de résultats doit donc être possible.

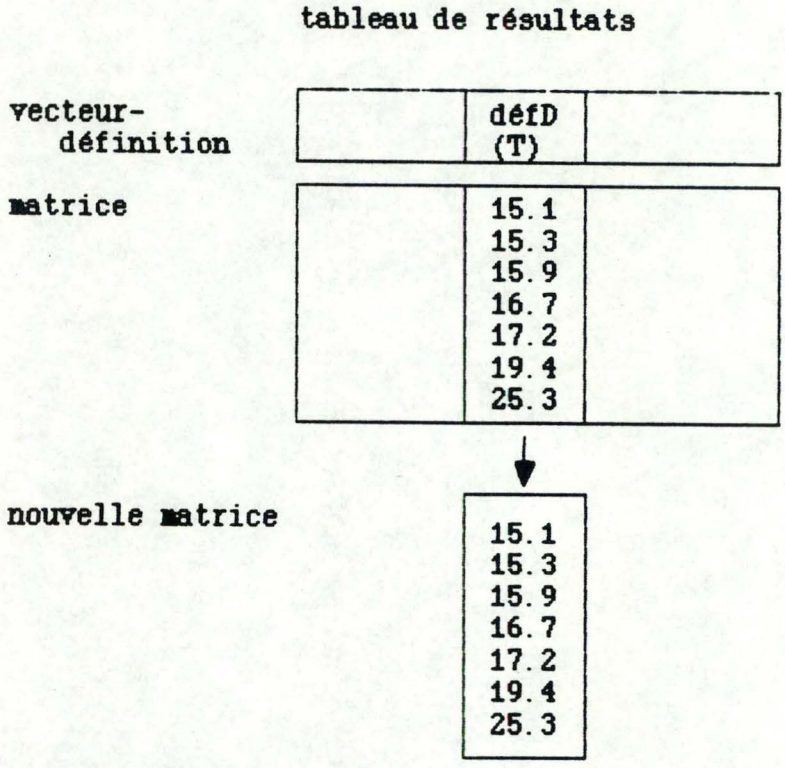
Ces différents outils de génération et de manipulation de matrices seront analysés en détail plus tard (voir manipulation de tableaux de résultats).

Quant à la description de l'état initial du système, les valeurs de départ à affecter aux variables évolutives et indépendantes non connues a priori sont données explicitement par l'utilisateur, lorsqu'il définit ces variables. Ces valeurs sont stockées dans les informations associées au genre de la variable (dans la définition de la variable).



Produit cartésien ordonné de la matrice A par la matrice B

figure 2.11.



Création d'une nouvelle matrice par extraction d'une colonne d'une matrice existante

figure 2.12.

2.7 Simulateur

Un simulateur est un programme, au sens informatique du terme, qui réalise sur base d'un calculateur, une série de simulations d'observation constituant une expérience simulée.

La série d'observations à simuler est planifiée quant aux variables indépendantes connues a priori grâce au schéma expérimental. Rappelons que celui-ci contient également, le cas échéant, la description de l'état initial du système. Par conséquent, un schéma expérimental préalablement constitué par l'utilisateur est associé à chaque exécution du simulateur. Toutes les valeurs des variables indépendantes connues a priori doivent être inscrites dans la matrice faisant partie du tableau de résultats fourni en entrée au simulateur.

Rappelons qu'un calculateur reçoit en entrée un vecteur de variables (dépendantes, indépendantes, ou évolutives) dans lequel les valeurs des variables indépendantes et évolutives sont connues. C'est le simulateur qui gère une série d'exécutions d'un calculateur en lui fournissant le vecteur adéquat à chacune de ses exécutions.

Le simulateur reçoit en entrée un tableau de résultats dont la matrice contient la partie du schéma expérimental correspondant aux variables indépendantes connues a priori. Il fournit en sortie le même tableau dont la matrice est mise à jour, c'est-à-dire où les valeurs des variables indépendantes connues a priori (schéma expérimental) sont intactes et celles des variables dépendantes, évolutives et indépendantes non connues a priori sont calculées (figure 2.13).

Il est à noter que le vecteur-définition du tableau de résultats fourni en entrée au simulateur doit avoir été rempli au préalable. Le vecteur doit contenir autant de définitions qu'il y a de paramètres au calculateur, et ne peut contenir la définition de deux variables portant le même nom.

Les valeurs des variables indépendantes non connues a priori sont calculées par le simulateur avant d'être fournies au calculateur. Le simulateur calcule ces valeurs selon des règles de calcul qui lui auront été précisées. Ces règles spécifient à quelle(s) variable(s) (variable(s)-référence) et ,pour chacune des variables-référence, à

quelle observation antérieure le simulateur doit se référer, ainsi qu'éventuellement les calculs à effectuer à partir des variables-référence.

tableau de résultats

	défI	défD	défE	défI	défE
vecteur-définition					
matrice	10	.	.	0.5	.
	20	.	.	0.75	.
	30	.	.	0.82	.
	40	.	.	0.93	.

SIMULATEUR

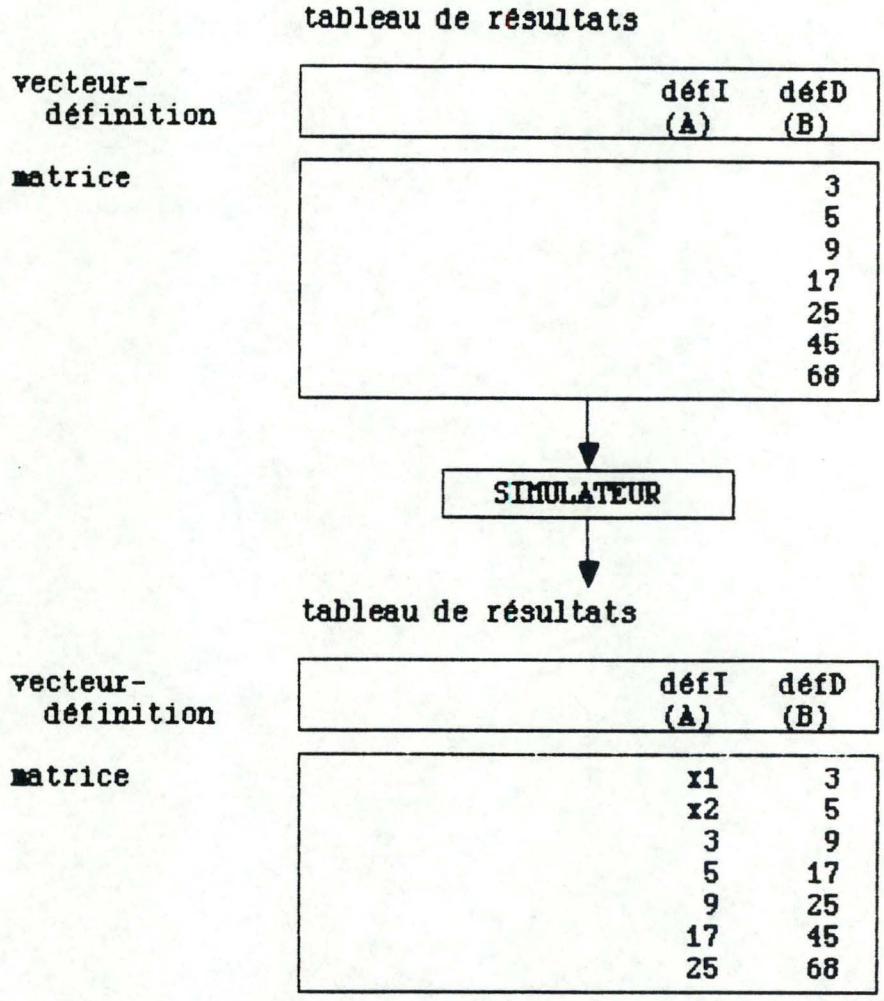
	défI	défD	défE	défI	défE
vecteur-définition					
matrice	10	-4	7	0.5	113.1
	20	-3.5	18	0.75	116.2
	30	-3.3	25	0.82	117.3
	40	-3.35	33	0.93	119.5

défI = définition de variable indépendante
défD = définition de variable dépendante
défE = définition de variable évolutive
. = valeur non significative

figure 2.13.

Remarquons qu'il faut encore préciser les valeurs à affecter aux variables indépendantes non connues a priori, lors des premières exécutions du calculateur au sein d'une expérience simulée. Appelons-les valeurs de départ. Ces valeurs font partie du schéma expérimental.

Supposons par exemple que l'on désire calculer la valeur d'une variable indépendante non connue a priori (soit A) sur base de la valeur d'une variable-référence (soit B) prise deux exécutions d'observation plus tôt. Cela signifie que les deux premières valeurs d'une telle variable ne peuvent être calculées. Il faut donc que l'expérimentateur donne explicitement ces deux premières valeurs (voir figure 2.14 où x1 et x2 sont les valeurs de départ).



défI = définition de variable indépendante
(ici non connue a priori)
défD = définition de variable dépendante

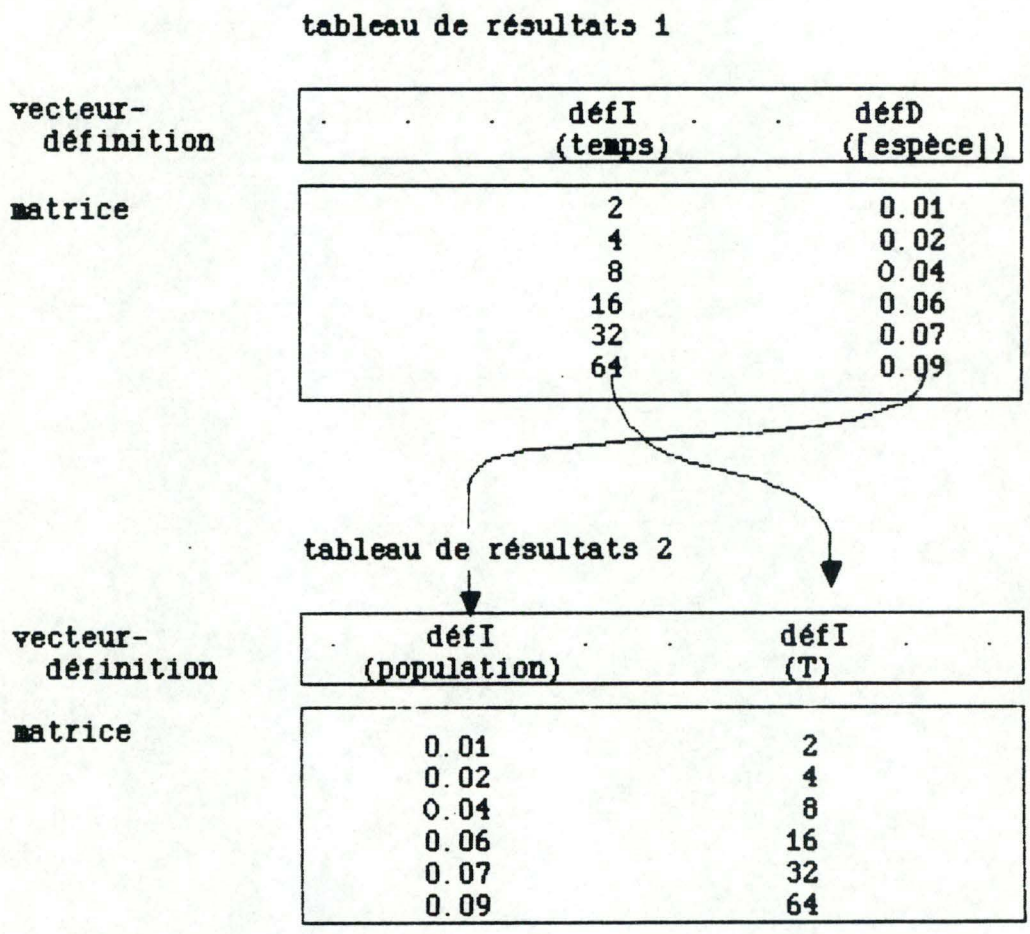
Le simulateur calcule les valeurs de la variable A à partir de la valeur que la variable-référence B prend deux exécutions d'observation plus tôt.

figure 2.14.

Le mode de détermination des variables indépendantes, ainsi que les règles de calcul et les valeurs de départ des variables indépendantes non connues a priori sont reprises dans les informations au sein de la définition de ces variables (voir implémentation du vecteur-définition, p.88).

D'autre part, la constitution d'un schéma expérimental peut, comme nous l'avons déjà dit, passer par la construction d'une matrice contenant les valeurs des variables indépendantes connues a priori. Nous avons fait remarquer à ce propos qu'il serait intéressant de pouvoir constituer une matrice en reprenant les valeurs stockées dans une colonne d'une matrice déjà existante.

Dans cette optique, il est nécessaire de prévoir un interfaçage entre différents simulateurs. Les résultats obtenus lors de l'exécution d'un simulateur peuvent ainsi être accessibles à l'expérimentateur qui désire les utiliser dans le but de définir le schéma expérimental d'un autre simulateur (voir figure 2.15).



défI = définition de variable indépendante
défD = définition de variable dépendante

figure 2.15.

CHAPITRE 3 : MANIPULATION DE TABLEAUX DE RESULTATS

3.1 Introduction

Rappelons qu'un tableau de résultats est constitué d'une matrice et d'un vecteur-définition.

La matrice est un tableau à deux dimensions dont chaque ligne contient les valeurs prises par les différentes variables après une observation simulée; l'ensemble des lignes de la matrice représente une expérience simulée. Chaque colonne de la matrice contient les valeurs prises par une même variable au cours de l'expérience simulée.

Le vecteur-définition reprend les caractéristiques de chaque variable: le nom, l'unité, le genre et des informations dépendant du genre. Ces caractéristiques constituent la définition de la variable.

Lors de la construction de nouveaux tableaux de résultats, l'utilisateur doit donc constituer un nouveau vecteur-définition et une nouvelle matrice. Mais il est possible que l'utilisateur désire constituer un nouveau tableau de résultats à partir de tableaux de résultats préexistants, en manipulant ceux-ci. En d'autres termes, il peut créer un nouveau vecteur-définition en reprenant des définitions de variables provenant de vecteurs-définition préexistants. Il peut aussi construiure une nouvelle matrice en recopiant des colonnes de valeurs stockées dans des matrices préexistantes.

Il serait également intéressant que le constructeur du didacticiel puisse construire sa nouvelle matrice en s'inspirant de tableaux de résultats différents de ceux dont il s'inspire pour constituer son vecteur-définition.

Nous pouvons illustrer ceci de la manière suivante. Supposons que l'expérimentateur désire constituer le vecteur-définition du tableau de résultats A en reprenant certaines définitions provenant des tableaux de résultats I et J. Il serait intéressant de lui permettre de construire

la matrice de A par extraction de colonnes de valeurs de matrices faisant partie des tableaux de résultats E, F et G.

Dans cette optique, la création d'une matrice est indépendante de la constitution du vecteur-définition faisant partie du même tableau de résultats. Nous choisissons de considérer le vecteur-définition et la matrice provenant d'un même tableau de résultats comme des entités de création et de manipulation différentes, car cette perspective nous paraît moins contraignante pour l'utilisateur. En effet, si nous ne dissocions pas les valeurs d'une variable de sa définition, le constructeur du didacticiel ne pourrait recopier les valeurs d'une variable sans reprendre sa définition.

Bien que les vecteurs-définition et les matrices soient considérés comme des entités de création et de manipulation séparées, aucune de ces entités n'a de signification si elle est isolée du tableau de résultats dont elle fait partie. C'est pourquoi, ces entités seront identifiées par un nom de tableau de résultats. Comme nous le verrons, si une primitive travaille sur une matrice ou un vecteur-définition particulier, ce sera le nom du tableau de résultats correspondant qui sera passé en argument à cette primitive.

Il est évident que pour exploiter un tableau de résultats, un certain nombre de contraintes doivent être respectées. Citons, à titre d'exemple, l'exploitation d'un tableau de résultats par un simulateur. Les conditions à respecter lors de l'exploitation d'un tableau de résultats par un simulateur sont les suivantes. Le vecteur-définition ne peut ni être vide, ni contenir deux variables ou plus de même nom. Les colonnes correspondant, dans une matrice, à des variables indépendantes connues a priori ne peuvent être vides. Enfin, le nombre de variables définies dans le vecteur-définition doit être égal au nombre de paramètres du calculateur.

A une étape donnée de son cycle de vie, il se peut que un tableau de résultats ne réponde pas dans son ensemble aux conditions requises par une exploitation particulière. Le tableau est alors dans un état intermédiaire, c'est-à-dire inexploitable (dans le cadre de l'exploitation envisagée).

Dans la suite de cette section, chaque fois que l'on parlera de matrice ou de vecteur-définition, il s'agira de matrice ou de vecteur-définition faisant partie d'un tableaux de résultats.

Nous parlerons également d'un vecteur-définition et d'une matrice associés ou correspondants. Il s'agira en fait d'un vecteur-définition et d'une matrice faisant partie du même tableau de résultats.

3.2 Manipulations de matrices

Le constructeur du didacticiel peut constituer un schéma expérimental à partir de matrice(s) préexistante(s), regrouper des résultats provenant de plusieurs simulateurs pour les visualiser ensemble, ou encore calculer des statistiques à partir de ces résultats. Nous proposons par conséquent à l'utilisateur des outils de manipulation de matrices.

Les différentes primitives de manipulation de matrices que nous proposons ci-dessous travaillent sur des matrices faisant partie de tableaux de résultats. C'est pourquoi les matrices passées en argument à ces primitives seront désignées par les noms des tableaux de résultats dont elles proviennent. Les vecteurs-définition appartenant aux tableaux de résultats passés en arguments ne sont pas modifiés par l'application de ces primitives.

Certaines conditions devront être satisfaites avant toute application de primitive travaillant sur des matrices. La (ou les) matrice(s) que l'on désire construire doit (doivent) être vide(s) avant l'application de l'outil. En d'autres termes, une future matrice ne peut contenir aucune valeur. Par contre, le (ou les) matrices(s)- que nous appelerons matrice(s) originale(s)- dont s'inspire l'outil de manipulation pour construire une nouvelle matrice ne peut (peuvent) être vide(s).

En général, si une des conditions requises par une primitive n'est pas satisfaite, la construction de la nouvelle matrice n'a pas lieu, et un message d'erreur est envoyé à l'utilisateur (par la primitive elle-même).

3.2.1 Création de matrice

La création de matrices contenant une série de valeurs (réels) pour chaque variable indépendante connue a priori est à la base de la construction de schémas expérimentaux. Il est donc nécessaire de disposer d'outils permettant de construire des matrices contenant des réels.

Les nombres à inscrire dans ces matrices peuvent être déterminés par l'expérimentateur, ou être la copie de valeurs se trouvant dans des colonnes de matrice(s) préexistante(s).

En ce qui concerne les outils de création de matrices dont les valeurs sont déterminées par l'expérimentateur, nous n'envisagerons ici que des primitives de création de matrice à une colonne (ou matrice-colonne). Cette option a été choisie en vue d'éviter une certaine lourdeur de manipulation qui serait imposée à l'utilisateur si ces primitives créaient en une seule fois, des matrices à plus d'une colonne.

Il paraît intéressant d'envisager au moins les primitives suivantes:

-génération d'une matrice-colonne contenant un nombre donné n (précisé par l'utilisateur) de réels générés aléatoirement (suivant une distribution connue: Poisson,...). Cette matrice contiendra n lignes.

-génération d'une matrice-colonne contenant (dans l'ordre) tous les nombres (réels) générés à partir d'une limite inférieure, par progression donnée, jusqu'à une limite supérieure; les limites et la progression sont précisés par l'utilisateur. Cette matrice contiendra autant de lignes que de nombres ainsi générés (voir figure 3.1).

-construction d'une matrice-colonne en donnant explicitement toutes les valeurs (réels) à inscrire dans la nouvelle matrice.

borne inférieure = 3.9
borne supérieure = 14.7
progression = 3.2

matrice

3.9
7.1
10.3
13.5

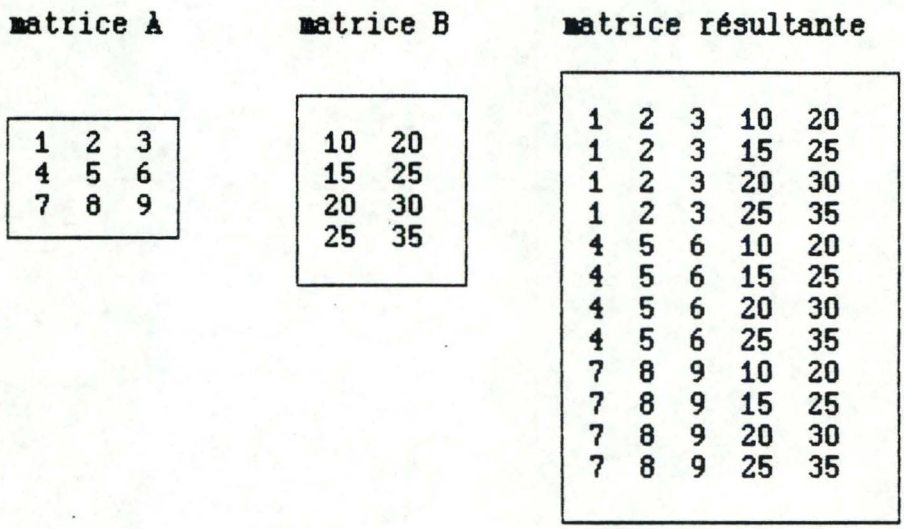
figure 3.1

La liste des outils de construction de matrices-colonne présentée ci-dessus n'est pas exhaustive.

3.2.2 Produit cartésien ordonné de matrices

Ce type de manipulation de matrice a déjà été évoqué à propos de la construction de schémas expérimentaux.

Cet outil effectue le produit cartésien ordonné d'une première matrice (A) à i lignes et p colonnes par une seconde matrice (B) à j lignes et q colonnes, et aboutit à la construction d'une nouvelle matrice à (i*j) lignes et (p+q) colonnes (voir figure 3.2).



produit cartésien ordonné de deux matrices

figure 3.2

Les valeurs de la matrice A sont copiées dans les p premières colonnes de la nouvelle matrice, et les valeurs de la matrice B dans les q colonnes suivantes. Chaque ligne de la matrice A est recopiée J fois, et chaque ligne de la matrice B l'est i fois. Les j copies d'une même ligne de la matrice A sont consécutives dans la nouvelle matrice. L'ordre des colonnes et des lignes n'est pas altéré lors de la copie des valeurs des matrices A et B.

3.2.3 Juxtaposition de matrices

Supposons un simulateur basé sur un calculateur utilisant un générateur de nombres aléatoires pour calculer les valeurs de ses variables dépendantes. Plusieurs exécutions de ce simulateur n'aboutiront pas aux mêmes résultats; les matrices seront donc différentes. L'expérimentateur peut désirer visualiser sur un même graphe une même relation entre deux variables mais provenant de matrices obtenues par différentes exécutions d'un même simulateur.

D'autre part, un même phénomène biologique observé et analysé peut donner lieu à plus d'un modèle mathématique. En outre, un même modèle mathématique peut, comme cela a déjà été dit, donner naissance à plus d'un calculateur, et donc à plus d'un simulateur.

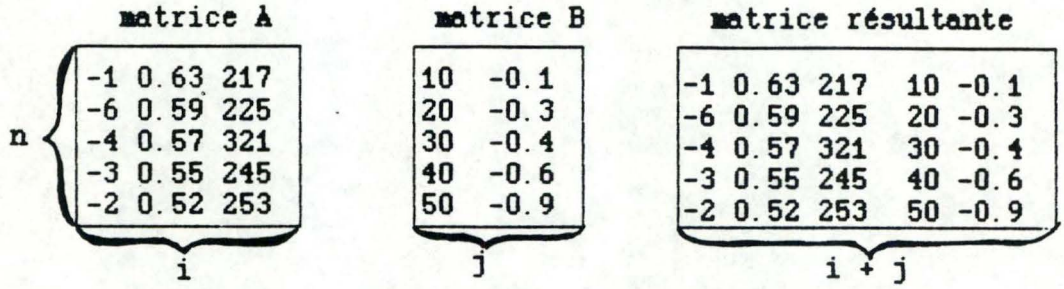
Ces exemples montrent que des simulateurs différents peuvent simuler un même phénomène biologique. Le constructeur du didacticiel peut trouver intéressant de visualiser sur un même graphe une relation entre deux variables observées dans les mêmes circonstances mais qui seraient simulées par exécution de différents simulateurs.

De plus, comme nous l'avons déjà vu, un schéma expérimental peut être constitué en partie par création de différentes matrices et par juxtaposition de ces matrices.

D'où l'intérêt de la primitive définie comme suit.

Nous appelerons juxtaposition de deux matrices données contenant toutes deux le même nombre de lignes, la copie des colonnes de la seconde matrice à la droite (d'une copie) des colonnes de la première matrice, et aboutissant à la construction d'une nouvelle matrice, tout en conservant l'ordre des colonnes des matrices originales (voir figure 3.3).

Si les matrices originales ne contiennent pas le même nombre de lignes , la nouvelle matrice n'est pas constituée, et un message d'erreur préviendra l'utilisateur.



juxtaposition de deux matrices de n lignes

la matrice A contient i colonnes
la matrice B contient j colonnes
la matrice résultante contient i + j colonnes

figure 3.3

3.2.4 Sélection de matrice

Un schéma expérimental peut être constitué en partie par les résultats obtenus lors de simulations d'expérience précédentes (voir schéma expérimental).

Il est également intéressant de pouvoir visualiser sous forme graphique une relation entre deux variables (soient A et B) seulement lorsqu'une troisième variable (soit C) répond à certains critères : par exemple lorsque C est égale, supérieure ou inférieure à une constante, ou est comprise entre deux bornes données. Ces quelques critères ne constituent pas une liste exhaustive des critères qui peuvent être envisagés.

Il est donc intéressant de pouvoir manipuler des matrices pour en extraire des informations intéressantes et construire une nouvelle matrice.

Nous appelerons sélection de colonnes de matrice l'extraction, à partir d'une matrice donnée, de colonnes spécifiées par un nom de variable (chaîne de caractères non vide), et la constitution d'une nouvelle matrice, constituée par la copie des colonnes extraites (voir figure 3.4). Ici aussi, l'ordre des colonnes et des lignes de la matrice originale est respecté dans la nouvelle matrice.

Temp	press	conc A	conc B
30	345	0.76	- 12
32	324	0.44	- 37
28	365	0.73	- 54
24	333	0.26	- 72
31	344	0.14	- 30
23	361	0.42	- 91

matrice résultante

30	345
32	324
28	365
24	333
31	344
23	361

création d'une nouvelle matrice par sélection des colonnes correspondant aux variables dont les noms sont Temp et press

figure 3.4

Nous appelerons sélection de lignes d'une matrice l'extraction, à partir d'une matrice donnée, de lignes sélectionnées sur base d'un critère, et la constitution d'une nouvelle matrice, constituée par la copie des lignes extraites. L'ordre des colonnes et des lignes de la matrice originale est respecté dans les nouvelles matrices.

Un certain nombre de conditions supplémentaires portant sur les arguments de la primitive doivent être satisfaites. S'il désire sélectionner des lignes dans une matrice, l'utilisateur devra préciser sur quelle variable (identifiée par un nom) porte la sélection, et sur base de quel critère. Le vecteur-définition du tableau de résultats sur lequel porte la sélection ne peut être vide ni contenir deux variables de même nom. Les variables sélectionnées sont identifiées par leur nom (chaîne de caractères non vide) et doivent correspondre à des variables définies dans ce vecteur-définition. Ce dernier doit contenir autant de définitions de variable qu'il y a de colonnes dans la matrice associée.

3.2.5 Duplication de matrice

Dans un schéma expérimental, lorsque deux variables indépendantes connues a priori prennent rigoureusement les mêmes valeurs tout au long d'une expérience simulée, il est intéressant de pouvoir constituer une matrice à une colonne et de la dupliquer facilement. En général, la duplication d'une matrice **à plus d'une** colonne semble intéressante lors de la construction d'un schéma expérimental.

Nous appelerons duplication d'une matrice la production d'une copie conforme d'une matrice donnée.

3.2.6 Statistiques sur une matrice

Le constructeur du didacticiel peut désirer exploiter les résultats d'un simulateur à des fins statistiques: calcul de moyenne, de variance,... Nous nous intéresserons ici à des statistiques portant sur une seule variable à la fois, par conséquent, de statistiques sur une colonne d'une matrice.

Nous appelerons statistiques sur une matrice le calcul de statistiques (moyenne, variance,...) sur les valeurs reprises dans chaque colonne d'une matrice donnée. Les statistiques ainsi calculées sont reprises dans une matrice à une seule ligne, appelons-la matrice statistique, dont le ième élément est le résultat de la statistique calculée sur les valeurs de la ième colonne de la matrice. Une telle matrice contient autant d'éléments qu'il y a de colonnes dans la matrice originale (voir figure 3.5).

La matrice statistique n'est autre qu'une matrice particulière, et constitue la matrice d'un tableau de résultats. Elle est par conséquent identifiée par le nom du tableau de résultats dont elle fait partie.

Cet outil statistique peut être considéré comme un opérateur particulier qui condense de l'information, à partir d'informations plus nombreuses reprises dans une matrice.

Tableau de résultats A contenant la matrice :

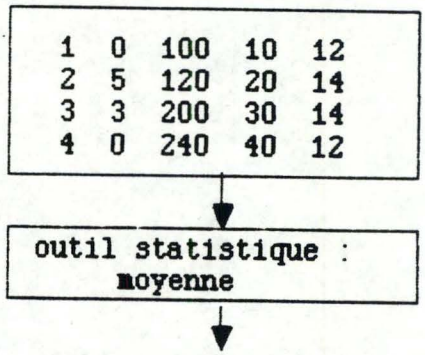


Tableau de résultats B contenant la matrice statistique :

2.5	2	165	25	13

figure 3.5

3.2.7 Superposition de matrices

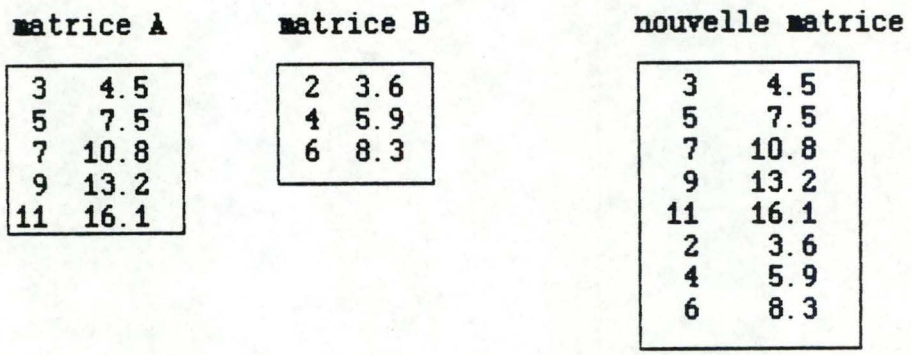
L'application d'un opérateur statistique à une matrice aboutit à la constitution de la matrice statistique. Les informations contenues dans la matrice statistique peuvent être visualisées à leur tour sous forme graphique. En effet, l'expérimentateur peut désirer regrouper les informations statistiques provenant de diverses matrices contenant les mêmes variables. Il s'agit en fait de rassembler plusieurs matrices statistiques, et de constituer ainsi une nouvelle matrice.

Il est possible aussi que l'expérimentateur désire visualiser à l'écran, sous forme d'un tableau, les résultats repris dans une matrice suivis directement du calcul des moyennes ou des variances (sur chaque colonne).

D'autre part, lors de la construction d'un schéma expérimental, une nouvelle matrice pourrait être constituée en reprenant dans sa partie supérieure des lignes d'une première matrice, et dans sa partie inférieure les lignes d'une autre matrice.

Cette manipulation peut également être utile dans le cas où l'utilisateur désire visualiser sur un même graphe une relation entre deux variables, en tenant compte des valeurs de cette relation reprises dans deux matrices différentes.

Nous appelerons superposition de deux matrices, respectivement de n et m lignes mais contenant toutes deux le même nombre de colonnes, la constitution d'une nouvelle matrice à n + m lignes dont les n premières lignes sont une copie des n lignes de la première matrice donnée , et les m dernières lignes sont une copie des m lignes de la seconde matrice donnée, tout en respectant l'ordre des lignes des matrices originales (figure 3.6).

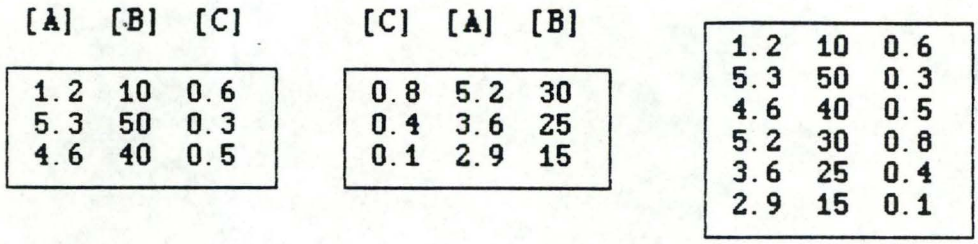


superposition de deux matrices

figure 3.6

Une option de ce genre de manipulation peut être envisagée. Cette option consiste à construire une nouvelle matrice par superposition de matrices provenant de deux tableaux de résultats dont les vecteurs-définition reprennent les définitions de variables de même nom mais éventuellement dans un ordre différent. Cette primitive pourrait porter le nom de superposition avec réarrangement.

Après l'application d'une telle manipulation, les valeurs correspondant à une même variable sont reprises dans une même colonne de la nouvelle matrice (voir figure 3.7).



superposition avec réarrangement

figure 3.7

Il faut qu'avant l'application de cette primitive le nombre de variables définies dans les vecteurs-définitions correspondant aux matrices à superposer soient égaux et corresponde au nombre de colonnes dans les matrices asociées. Il faut également que les variables définies dans le premier vecteur-définition portent les mêmes noms que celles définies dans le second vecteur-définition.

3.2.8 Suppression de matrice

Cet outil permet d'effacer toute trace des valeurs reprises dans une matrice. Ce qui équivaut à remettre "à vide" cette matrice.

Cette primitive est nécessaire si l'on désire pouvoir réutiliser des tableaux existants pour construire des nouvelles matrices. Rappelons que toute future matrice doit être vide avant l'application d'une primitive de construction de matrice.

Cet outil sera aussi particulièrement utile, dans un souci d'économie de place mémoire, notamment dans le cas où le constructeur de didactciel manipule un nombre élevé de tableaux de résultats, et par conséquent de matrices.

3.2.9 Tri d'une matrice

Supposons que l'on désire visualiser ensemble les résultats d'expériences repris dans deux tableaux de résultats différents, mais qui partagent une même variable, c'est-à-dire une variable qui a la même définition. Supposons aussi que les valeurs de la variable commune soient identiques dans les deux matrices, mais n'apparaissent pas dans le même ordre. Il serait intéressant de pouvoir retrier les lignes de chacune des matrices de sorte que les valeurs de la variable commune apparaissent dans un ordre croissant (ou décroissant).

La visualisation, sous forme graphique, d'une relation entre deux variables requiert dans certains cas un tri des valeurs des deux variables sur base d'une des deux variables (voir concepts de graphe).

C'est pour ces raisons que nous proposons l'outil suivant.

Nous appelerons tri croissant (décroissant) d'une matrice donnée, la construction d'une nouvelle matrice constituée des mêmes lignes et mêmes colonnes que la matrice donnée, mais où les lignes sont réarrangées de sorte que les valeurs d'une variable donnée apparaissent dans un ordre croissant (décroissant). En cas d'égalité de valeurs de cette variable, l'ordre des lignes dans lesquelles la variable a la même valeur sera identique à l'ordre d'apparition de ces lignes dans la matrice originale (voir figure 3.8).

Temp

5.2	30	108
6.7	25	97
3.9	18	89
1.2	31	101
0.7	27	98
4.3	25	92

3.9	18	89
6.7	25	97
4.3	25	92
0.7	27	98
5.2	30	108
1.2	31	101

tri croissant sur la variable de nom Temp

figure 3.8

La variable sur laquelle se base le tri est identifiée par son nom (chaîne de caractère non vide). Il faut qu'une variable de même nom ait été définie dans le vecteur-définition original.

3.3 Manipulations de vecteurs-définition

L'utilisateur peut créer un nouveau vecteur-définition en spécifiant individuellement chaque définition de variable. Mais comme pour les matrices, le vecteur-définition peut être constitué à partir de vecteur(s)-définition provenant de tableau(x) de résultat(s) préexistant(s).

Les vecteurs-définition sur base desquels un nouveau vecteur est construit (ainsi que ce nouveau vecteur) et qui sont passés en arguments aux primitives de construction, sont désignés par les noms des tableaux de résultats dont ils proviennent. Il est évident que ces primitives ne travailleront que sur les vecteurs-défintion de ces tableaux de résultats.

Les matrices des différents tableaux de résultats passés en arguments à ces primitives ne sont en rien modifiées par l'application d'outils de manipulation de vecteurs-définition.

Les nouveaux vecteurs-définition sont constitués uniquement s'ils ne comprennent pas deux variables de même nom. De même, les vecteurs-définition originaux ne peuvent contenir deux variables de même nom. Ce choix se justifie par le fait que le nom d'une variable est identifiant au sein d'une même tableau.

En général, les primitives travaillant sur des vecteurs-définition exigent que des conditions portant sur des arguments soient satisfaites. Si une des conditions requises n'est pas remplies, le nouveau vecteur-définition n'est pas constitué et un message d'erreur est affiché (par la primitive elle-même).

3.3.1 Ajout d'une définition

Un nouveau tableau de résultats peut être construit sans être le fruit d'une manipulation de tableaux de résultats préexistants. C'est pourquoi, il faut offrir à l'utilisateur la possibilité de définir de nouveaux vecteurs-définition autrement que par manipulation de vecteur(s)-définition provenant de tableaux de résultats préexistants.

Une telle primitive a pour effet d'ajouter à la fin d'un vecteur-définition donné la définition d'une variable donnée par l'utilisateur. Par conséquent, un vecteur-définition contenant i définitions, après l'application de cette primitive contiendra i + 1 définitions, la dernière étant la nouvelle définition donnée par l'utilisateur.

Cette primitive peut être utilisée même si le vecteur-définition est vide. Après l'application de la primitive, le vecteur-définition ne contiendra alors qu'une seule définition.

Le nom de la nouvelle variable (nouvelle définition) est une chaîne de caractères non vide.

3.3.2 Duplication de vecteur-définition

Nous appelerons duplication d'un vecteur-définition la construction d'un nouveau vecteur-définition qui est la copie conforme d'un vecteur-définition d'un tableau de résultats préexistant. Ce dernier vecteur ne peut être vide.

La primitive efface les éventuelles définitions de variables que pourrait déjà contenir le nouveau vecteur avant l'application de la primitive.

3.3.3 Juxtaposition de vecteurs-définition

Nous appelerons juxtaposition de vecteurs-définition la construction, à partir de deux vecteurs-définition donnés non vides, d'un nouveau vecteur-définition par juxtaposition d'une copie du second vecteur à la droite d'une copie du premier vecteur.

La primitive efface les éventuelles définitions de variables que pourrait déjà contenir le nouveau vecteur avant l'application de la primitive.

3.3.4 Sélection de vecteur-définition

Nous appelerons sélection de vecteur-définition la construction d'un nouveau vecteur-définition constitué d'une copie de la définition de(s) variable(s) sélectionnée(s) par l'utilisateur et identifiée(s) par leur nom (chaîne de caractères non vide). Ces définitions sont sélectionnées au sein d'un vecteur déjà existant mais non vide. L'ordre des définitions du nouveau vecteur correspond à l'ordre de ces définitions dans le vecteur original.

Si le vecteur-définition original ne contient pas la définition d'une variable de même nom qu'une variable sélectionnée, la sélection n'a pas lieu.

Cette primitive efface les éventuelles définitions de variable que pourrait déjà contenir le nouveau vecteur avant l'application de la primitive.

3.3.5 Modification d'une définition de variable

Nous appelerons modification d'une définition d'une variable le remplacement, au sein d'un vecteur-définition donné, de la définition d'une variable (spécifiée par son nom) par une nouvelle définition donnée par l'utilisateur.

Le nom de la variable dont la définition doit être remplacée ne peut être une chaîne de caractères vide. Il en va de même pour le nom de la variable de la nouvelle définition.

3.4 Manipulations globales de tableaux de résultats

La matrice et le vecteur-définition d'un nouveau tableau de résultats peuvent être constitués à partir du même tableau de résultats. La définition d'une nouvelle variable (vecteur-définition) et des valeurs que celle-ci prend au cours d'une expérience simulée (matrice) sont alors la copie de la définition et des valeurs d'une variable d'un tableau de résultats donné. Les valeurs d'une variable

reprises dans une matrice ne sont donc pas considérées ici indépendamment de la définition de cette variable.

La création de la matrice et du vecteur-définition d'un nouveau tableau de résultats se fera par conséquent en une seule étape. C'est donc dans le but d'augmenter encore la facilité de l'utilisateur que les outils analysés ci-dessous sont proposés.

Certaines conditions doivent être satisfaites avant l'application de ces différentes primitives. La matrice du nouveau tableau de résultats que l'on désire construire doit être vide avant l'application d'outils de manipulation de tableaux. Par contre, ni le vecteur-définition, ni la matrice des tableaux de résultats originaux ne peut être vide. Les vecteurs-définition originaux ne peuvent contenir la définition de deux variables portant le même nom. Le nouveau tableau de résultats sera constitué à condition que deux variables définies dans le nouveau vecteur-définition ne portent pas le même nom.

En général, si une des conditions requise par une primitive n'est pas satisfaite, la construction du nouveau tableau n'a pas lieu et un message d'erreur est envoyé à l'utilisateur par la primitive.

Les primitives suivantes effacent les éventuelles définitions de variables que pourrait déjà contenir le vecteur-définition du nouveau tableau de résultats avant l'application de la primitive.

3.4.1 Produit cartésien ordonné de tableaux

Nous appelerons produit cartésien ordonné de deux tableaux de résultats la construction d'un nouveau tableau dont la matrice est constituée par produit cartésien ordonné des deux matrices originales et le vecteur-définition par la juxtaposition des vecteurs originaux.

Ce produit cartésien de deux tableaux sera permis si et seulement s'il y a autant de colonnes dans les matrices originales que de variables définies dans les vecteurs-définition correspondants.

3.4.2 Juxtaposition globale de tableaux

Nous appelerons juxtaposition globale de deux tableaux dont les matrices contiennent toutes deux le même nombre de lignes, la construction d'un nouveau tableau de résultats dont la matrice est constituée par la juxtaposition des matrices originales et le vecteur-définition par la juxtaposition des vecteurs-définition originaux.

La juxtaposition de tableaux de résultats se fera correctement si les matrices originales contiennent le même nombre de lignes, et si il y a autant de colonnes dans les matrices originales que de variables définies dans les vecteurs-définition correspondant.

3.4.3 Sélection globale de tableaux

Nous appelerons sélection globale de tableaux de résultats, la constitution à partir d'un tableau de résultats donné d'un nouveau tableau dont la matrice est obtenue par sélection de lignes ou de colonnes (voir sélection de matrice) et dont le vecteur-définition reprend les définitions du vecteur original correspondant aux variables des colonnes sélectionnées (si la sélection porte sur des lignes, le nouveau vecteur une copie conforme du vecteur original).

Les variables sélectionnées le sont par leur nom (chaîne de caractères non vide). Si le vecteur-définition original ne contient pas la définition d'une variable de même nom qu'une variable sélectionnée, la sélection n'a pas lieu et le nouveau tableau de résultats n'est pas constitué.

La matrice originale doit contenir autant de colonnes qu'il y a de variables définies dans le vecteur-définition correspondant.

3.4.4 Duplication globale de tableau

Nous appelerons duplication de tableau de résultats la production d'une copie conforme d'un tableau de résultats donné (matrice et vecteur-définition).

3.4.5 Statistiques globales sur un tableau

Nous appelerons statistiques globales sur un tableau de résultats la constitution d'un nouveau tableau de résultats dont la matrice est obtenue par des calculs effectués sur la matrice originale (voir statistiques sur une matrice) et le vecteur-définition est une copie conforme du vecteur original.

La matrice sur laquelle est calculée la statistique doit contenir autant de colonnes qu'il y a de variables définies dans le vecteur-définition correspondant.

3.4.6 Superposition globale de tableaux

Nous appelerons superposition globale de deux tableaux de résultats dont les matrices contiennent le même nombre de colonnes, la constitution d'un nouveau tableau de résultats dont le vecteur-définition est une copie conforme du premier vecteur original, et dont chaque colonne de la matrice est constituée par superposition des colonnes des matrices originales correspondant à une même variable.

Les deux matrices originales doivent contenir le même nombre de colonnes, et les noms des variables définies dans les deux vecteurs-définition doivent être identiques. L'ordre des définitions des variables peut cependant être différent dans les deux vecteurs-définition; il s'agit alors de superposition avec réarrangement. Dans chaque tableau de résultats original, il doit y avoir autant de définitions de variables que de colonnes dans la matrice.

3.4.7 Tri d'un tableau

Nous appelerons tri croissant (décroissant) d'un tableau de résultats la construction d'un nouveau tableau de résultats dont le vecteur-définition est une copie conforme du vecteur-définition original, et la matrice est obtenue par tri croissant (décroissant) de la matrice originale sur base des valeurs d'une variable donnée (voir tri de matrice).

La variable sur laquelle se base le tri est identifiée par son nom (chaîne de caractères non vide). Une variable de même nom doit avoir été définie dans le vecteur-définition original.

CHAPITRE 4 : CONCEPTS RELATIFS AU GRAPHE

Les résultats obtenus par simulation et stockés dans des tableaux de résultats intéressent l'utilisateur final, c'est-à-dire l'élève. Le constructeur du didacticiel désire y avoir accès et les exploiter pour permettre à celui-ci de les visualiser. Il existe diverses manières de visualiser les informations contenues dans les tableaux de résultats. Une des manières les plus souvent utilisées en biologie, surtout dans le domaine de l'enseignement, consiste à illustrer une relation entre deux variables sous forme graphique. Dans un but didactique, cette façon de présenter les résultats paraît souvent plus efficace auprès des élèves.

Nous envisagerons, dans le cadre de ce travail, plusieurs façons de représenter graphiquement des résultats, que nous appellerons courbe, diagramme en bâtonnets et histogramme. Ces différents concepts, ainsi que ceux qui y sont associés, sont développés ci-dessous.

4.1 Fenêtre

Nous appelons fenêtre un cadre défini par les coordonnées du coin supérieur gauche (HG) et du coin inférieur droit (BD).

Un tel cadre peut être imprimé à l'écran; il doit toutefois être contenu dans les limites de l'écran. Il est possible d'imprimer quelque chose dans les limites de la fenêtre, ou encore d'effacer celle-ci.

La possibilité est offerte à l'utilisateur de définir plusieurs fenêtres et de les imprimer ensemble sur un même écran. Les fenêtres affichées peuvent se chevaucher. En cas de chevauchement, le contenu de la dernière fenêtre imprimée se superpose au contenu des autres fenêtres, et par conséquent l'efface. Les parties de fenêtre occultées par la superposition d'une nouvelle fenêtre sont définitivement inaccessibles.

Pour sa facilité, nous proposons à l'utilisateur de définir les coordonnées d'une fenêtre en terme de pourcent de la largeur et de la hauteur maximum de l'écran. Chaque coordonnée d'une fenêtre -HG ou BD- sera dorénavant identifiée par un couple de valeurs. La première valeur étant associée à la largeur de l'écran, la seconde à la hauteur de l'écran.

Si nous définissons les coordonnées d'une fenêtre en terme de pourcent des dimensions de l'écran, les coordonnées HG = (25, 25) et BD = (50, 50) ne définissent pas forcément une fenêtre carrée (voir figure 4.1).

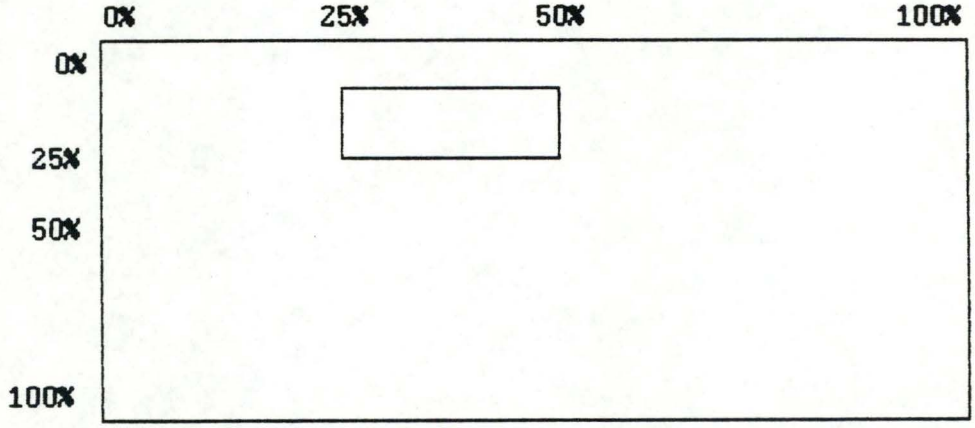
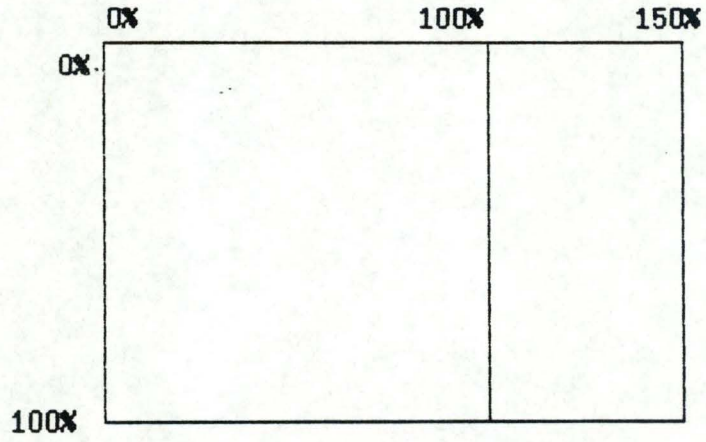


figure 4.1

Il paraît cependant intéressant de garder cette notion de pourcentage pour définir une fenêtre.

Nous conviendrons alors que le minimum de la largeur et de la hauteur de l'écran, mesurées dans une même unité de longueur, représente le pourcentage maxima, à savoir 100%. Les écrans sont en général plus large que haut, ce qui implique que ces 100% sont le plus souvent associés à la hauteur.

Dans cette optique, la fenêtre (0, 0) (100, 100) ((0, 0) correspondant à HG et (100, 100) à BD) est le plus grand carré affichable. Ceci n'exclu pas qu'une fenêtre plus grande puisse être affichée. Le rectangle (0, 0) (150, 100) peut être imprimé si la largeur de l'écran est au moins une fois et demi plus grande que la hauteur (voir figure 4.2).



hauteur = 24 cm
largeur = 36 cm

figure 4.2

Cette façon de déterminer les fenêtres assure la portabilité du didacticiel, si son constructeur travaille sur un écran fictif de 100 sur 100 (%). Dans ce cas, quelle que soit les dimensions réelles de l'écran utilisé, toute fenêtre inclue dans le carré (0,0) (100,100) est affichable.

Dans l'environnement matériel et logiciel où nous nous trouvons , les instructions graphiques ne sont encore guère normalisées. Nous voudrions cependant que le didacticiel soit portable le plus aisément possible. C'est dans cette intention que les concepts et conventions suivants ont été développés.

Dans beaucoup de cas , le seul système de coordonnées écran accessible est celui exprimé directement en pixels. Notre système devra donc lui-même transformer les coordonnées en % en nombre de pixels. Comme, d'autre part, la distance entre deux pixels successifs est différente selon que l'on se déplace horizontalement ou verticalement,

le système aura besoin d'informations supplémentaires. Nous conviendrons de définir les paramètres suivants :

-nbpxh : le nombre de pixels sur la longueur de l'écran.
-nbpxv : le nombre de pixels sur la hauteur de l'écran.
-aspect : rapport de la hauteur à la largeur de l'écran exprimées dans une même unité de longueur.

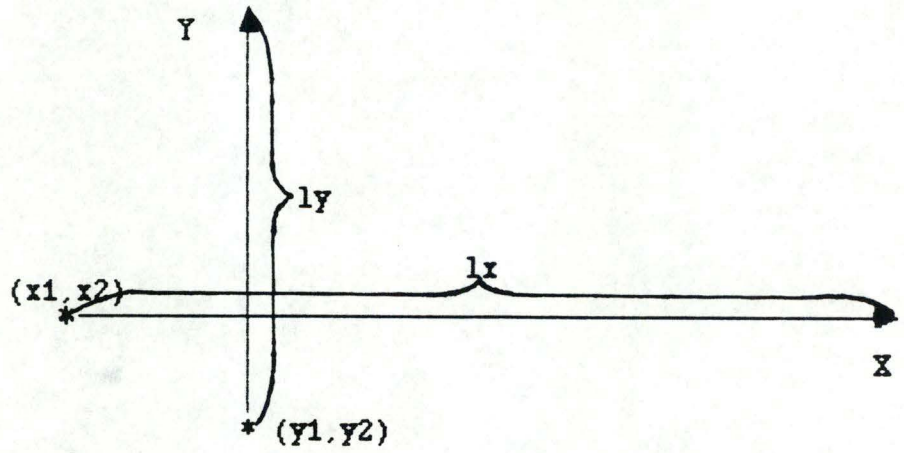
Ces paramètres sont également utiles dans le cas d'application où les notions d'angles, de circonférence, et en général, les propriétés euclidiennes des figures sont importantes.

4.2 Axe

L'axe des abscisses (ou axe X) est un segment de droite horizontal et est défini par les coordonnées d'un point de la fenêtre (que nous appelerons extrémité-origine de X) et une longueur . L'axe des ordonnées (ou axe Y) est un segment de droite vertical également défini par les coordonnées d'un point de la fenêtre (extrémité-origine de Y) et une longueur (voir figure 4.3).

Un système d'axes (c'est-à-dire un axe vertical et un axe horizontal) est défini dans une fenêtre. Par conséquent, pour la facilité de l'utilisateur, les coordonnées des extrémités-origines des axes seront données en terme de pourcent de la largeur et hauteur de la fenêtre dans laquelle ils doivent être tracés, et comptées à partir du coin inférieur gauche. La longueur est aussi exprimée en termes de pourcents des dimensions de la fenêtre. Les pourcentages maxima de la largeur et de la hauteur de la fenêtre sont de 100%, et les pourcentages minima sont affectés au coin inférieur gauche de la fenêtre (voir figure 4.4).

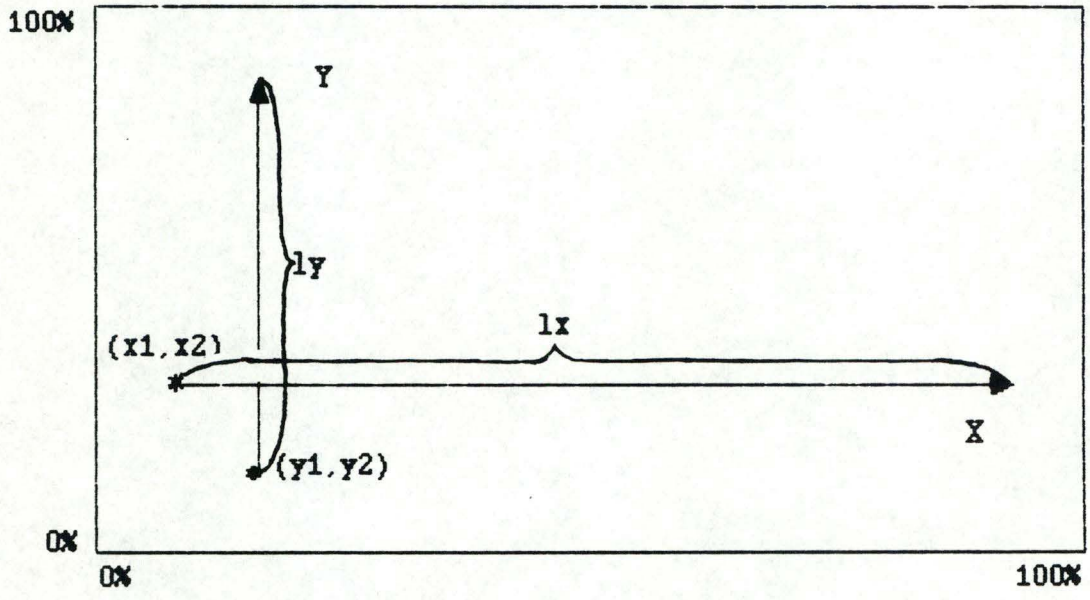
Cette manière de procéder permet une adaptation aisée à toute modification des dimensions de la fenêtre. De plus, dans toute fenêtre carrée, deux longueurs égales ont même mesure, qu'elles soient horizontales ou verticales.



origine de X = (x1,x2)
origine de Y = (y1,y2)
longueur de X = lx
longueur de Y = ly

L'axe de longueur lx est tracé horizontalement à partir de l'extrémité-origine de X.
L'axe de longueur ly est tracé verticalement (en remontant) à partir de l'extrémité-origine de Y.

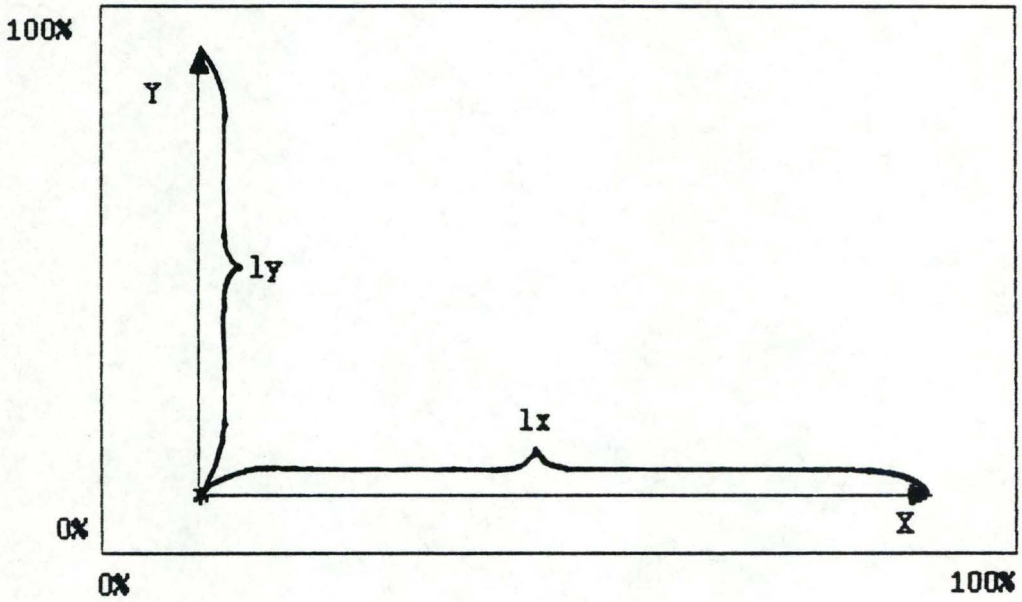
figure 4.3



(x1,x2) = (10,30)
(y1,y2) = (20,15)
lx = 80
ly = 70

figure 4.4

Les extrémités-origine des axes et leur longueur sont déterminées soit en utilisant l'option par défaut (voir figure 4.5), soit explicitement en précisant les coordonnées des extrémités-origines et les longueurs des axes.

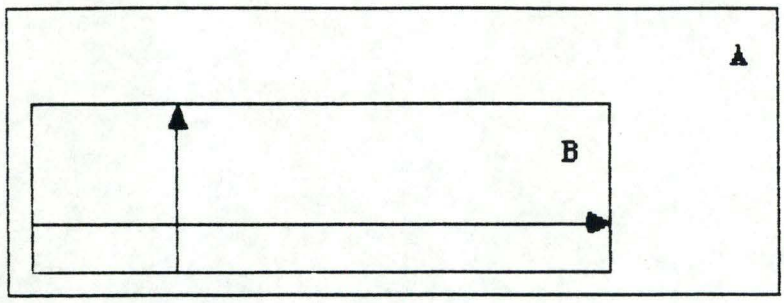


* = (x1,x2) = (y1,y2) = (10,10)
lx = ly = 80
(valeurs non définitives)

exemple d'option par défaut pour la définition d'un système d'axes au sein d'une fenêtre

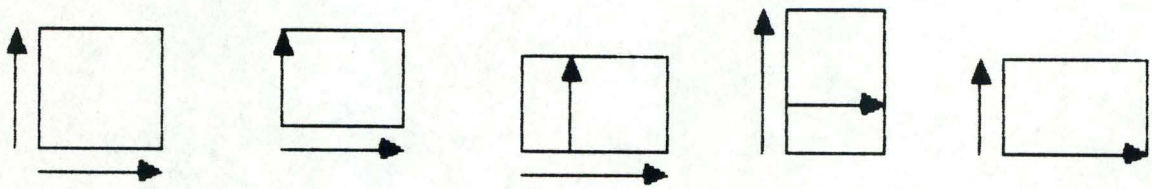
figure 4.5

Tout système d'axes détermine une zone rectangulaire de l'espace dans laquelle une relation entre deux variables est illustrée (voir figure 4.6). Cette zone que nous appelerons zone d'affichage a pour hauteur le segment constituant l'axe vertical, et pour largeur le segment constituant l'axe horizontal. La zone d'affichage peut ou non contenir les axes (voir figure 4.7). Dans tous les cas, les projections des points de la zone d'affichage doivent tomber sur les axes.



A = fenêtre
B = zone d'affichage

figure 4.6



exemples de zones d'affichage : 

figure 4.7

4.3 Coordonnées d'une relation

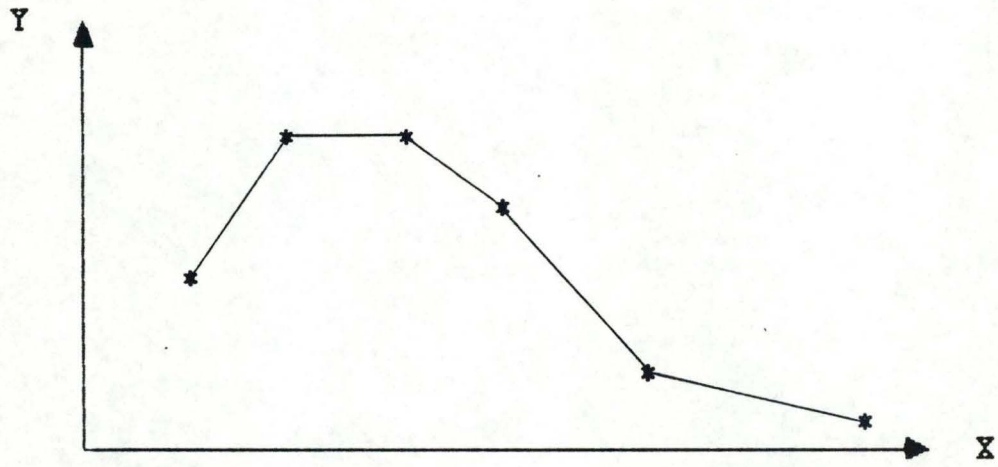
Nous appelerons coordonnées d'une relation l'ensemble des coordonnées des points (couple de valeurs) d'une relation à représenter dans une zone d'affichage.

Nous désirons pouvoir représenter toute relation entre deux variables d'un tableau de résultats. Une relation est donc déterminée par les valeurs reprises dans deux colonnes d'une même matrice. Pour déterminer une relation entre deux variables, il faut par conséquent préciser le nom d'un tableau de résultats, le nom de la variable à porter en abscisse ainsi que celui de la variable à porter en ordonnée.

Rappelons qu'un outil de manipulation de matrice (voir sélection de matrice) permet à l'utilisateur de construire une nouvelle matrice par sélection de lignes satisfaisant à certains critères. Ceci est utile dans le cas où l'utilisateur désire visualiser une relation entre deux variables (A et B) seulement lorsqu'une troisième variable (C) satisfait certaines conditions, par exemple lorsque C est inférieure, égale ou supérieure à une valeur donnée, ou encore lorsque C est comprise entre deux bornes données.

4.4 Courbe

Nous appelerons courbe l'ensemble des points illustrant une relation entre deux variables sur un graphe. Chaque couple de points voisins (ou consécutifs) de cette relation peut être ou non, selon le choix de l'utilisateur, relié par un segment de droite que nous appelerons lien (voir figure 4.8).



courbe dont les points voisins sont reliés par une portion de droite ou lien

figure 4.8

Le fait de parler de points voisins ou consécutifs sous-entend la notion d'ordre. Dans bon nombre d'applications, l'ordre est relatif aux valeurs d'abscisses. Les valeurs de la variable portée en abscisse sont triées avant de tracer la courbe (revoir figure 4.8)

La notion d'ordre peut être associée, dans d'autres cas, aux valeurs d'ordonnées. Ce sont alors les valeurs de la variable portée en ordonnée que l'on trie au préalable (figure 4.9).

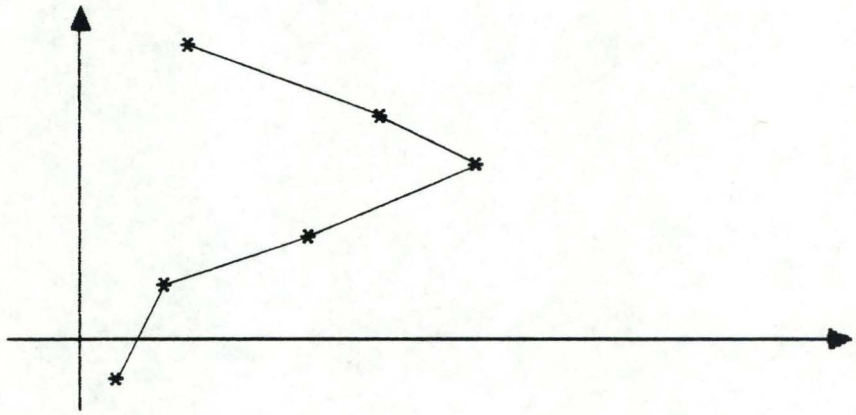


figure 4.9

Il serait également intéressant de pouvoir visualiser sur un même graphe la relation entre deux variables en fonction de l'évolution d'une troisième variable. Imaginons qu'un tableau de résultats contienne la définition d'une variable représentant le temps, et que les valeurs associées à cette variable apparaissent dans la matrice par ordre croissant. Supposons que l'on désire visualiser l'évolution dans le temps de la relation entre deux autres variables appartenant au même tableau de résultats. L'ordre des points de la relation correspond alors à l'ordre d'apparition des lignes dans la matrice (voir figure 4.10).

Nous laisserons donc à l'utilisateur le choix quant à la manière de faire les liens entre les points d'une relation. Ce peut être:

- dans l'ordre croissant des valeurs d'abscisse
- dans l'ordre croissant des valeurs d'ordonnée
- dans l'ordre des lignes du tableau de résultats

Dans beaucoup d'applications, une même variable conditionne l'évolution de plusieurs autres variables. Supposons que l'on observe l'évolution de la population de deux espèces animales en fonction de l'évolution de la population d'une troisième espèce, prédatrice des deux

premières. Il semble naturel de porter en abscisse la population du prédateur, et en ordonnée les populations des proies. Deux relations, ou courbes, différentes sont alors représentées par rapport au même système d'axe.

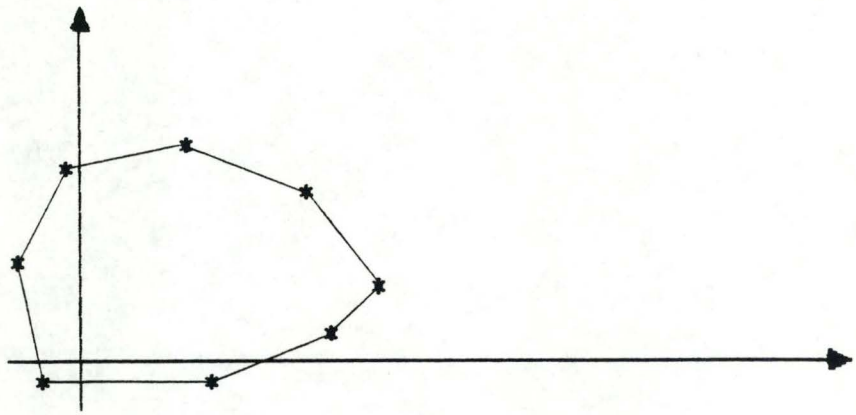


figure 4.10

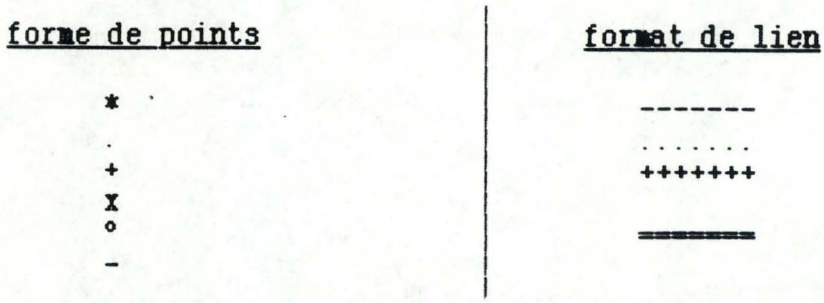
C'est pourquoi nous offrons à l'utilisateur la possibilité de tracer plusieurs courbes par rapport à un même système d'axes.

Pour chacune de ces courbes, il faut préciser les coordonnées de la courbe et le patron du tracé (cfr infra).

4.5 Patron du tracé d'une courbe

Nous appellerons patron du tracé d'une courbe la forme des points à dessiner ainsi que le format des liens (ou portion de droites) reliant les couples de points voisins d'une relation, dans le cas où l'utilisateur désire voir ces points reliés. On entend par format des liens, le type de trait reliant les deux points. Quelques formes de points et formats de liens sont présentés à la figure 4.11.

La forme des points et le format de liens peuvent être plus diversifiés si la machine dispose de la couleur. A chaque forme de point et format de lien proposé est associé un code. La correspondance entre un code et une forme de point ou un format de lien peut être établie ou modifiée par l'utilisateur lui-même, de sorte que si celui-ci change de machine, ce code peut être mis-à-jour en fonction des possibilités offertes par la machine.



présentation de quelques formes de points et formats de lien

figure 4.11

Tous les points d'une même courbe auront la même forme et les liens reliant deux points voisins d'une même courbe auront le même format. Pour chacune des courbes à tracer sur un graphe, le patron du tracé désiré est spécifié. La distinction entre les différentes courbes tracées par rapport à un même système d'axes est ainsi plus aisée pour l'observateur.

4.6 Titre de graphe, titre d'axe et unité d'axe

Le titre d'un graphe est une chaîne de caractères de longueur limitée qui est imprimée au dessus du système d'axes correspondant, mais en dehors de la zone d'affichage. Le titre d'un graphe peut être composé de plusieurs lignes à condition que l'utilisateur, en définissant un sytème d'axes au sein d'une fenêtre, ait prévu suffisamment de place. Ceci permet d'imprimer éventuellement quelques commentaires au dessus de la zone d'affichage. Le titre d'un graphe n'est pas obligatoire.

Le titre d'un axe et l'unité de l'axe sont deux chaînes de caractères de longueur limitée imprimées à proximité de l'axe correspondant, et précisant respectivement l'identité de l'axe et l'unité dans laquelle les valeurs imprimées sur cet axe sont exprimées (voir concept d'échelle). L'unité de l'axe n'est autre que l'unité reprise dans la définition de la variable associée à l'axe.

La longueur du titre de l'axe suivi de son unité (imprimée entre parenthèses) doit être inférieure à la largeur de la fenêtre.

Le titre et l'unité de l'axe X sont imprimés en dessous de cet axe, tandis que le titre et l'unité de l'axe Y sont imprimés au dessus de cet axe.

Un certain nombre d'autres indications sont imprimées également à proximité de celui-ci. Il s'agit des marques et repères d'étalonnage (cfr infra).

L'ensemble des renseignements accompagnant un axe seront placés dans la fenêtre près de l'axe. Si un des axes se trouve au bord de la zone d'affichage, et s'il est possible d'imprimer les dits renseignements dans la fenêtre mais en dehors de la zone d'affichage, le système les y placera automatiquement. Dans le cas contraire, les indications relatives à un axe seront plaçées dans la zone d'affichage. La priorité sera toutefois laissée à l'affichage des points des courbes. Les liens, par contre, seront moins prioritaires que les indications considérées (voir figure 4.12).

4.7 Fourchette

Après avoir défini une fenêtre et y avoir tracé un système d'axes, il faut encore déterminer les valeurs associées à chaque point des axes. Ceci introduit la notion de fourchette. Pour chacun des axes, l'utilisateur définit la valeur associée à l'intersection avec l'autre axe ainsi que celle associée à l'extrémité finale. Si les axes ne se recoupent pas, l'utilisateur précise les valeurs associées aux extrémités-origine et finales des axes.

Notons que l'extrémité-origine d'un axe ne correspond pas forcément à l'intersection avec l'autre axe. Ces deux points particuliers d'un axe ne doivent pas être confondus (voir figure 4.13).

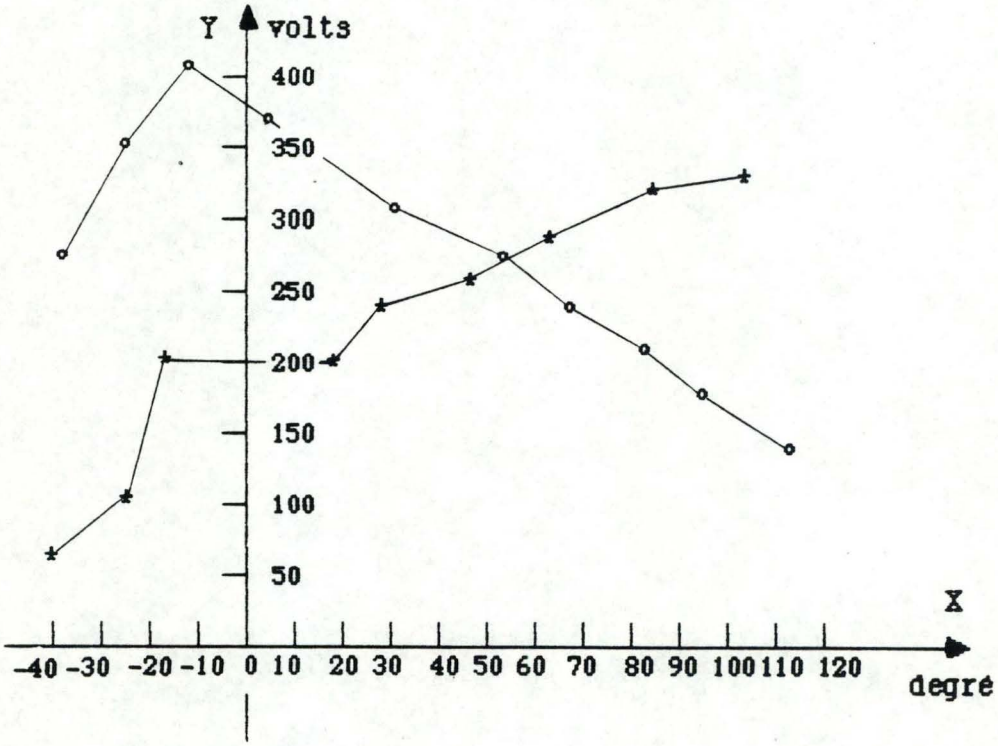
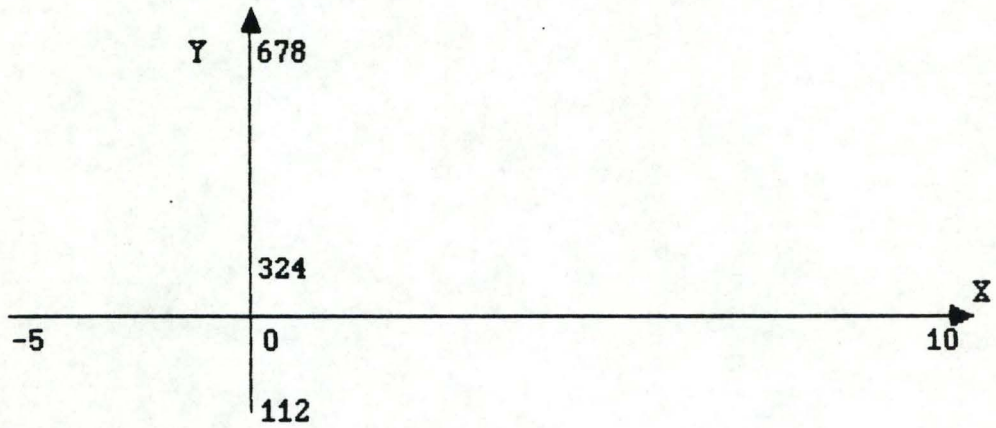


figure 4.12



Valeurs données par l'utilisateur pour définir:
- la fourchette de l'axe X = (0,10)
- la fourchette de l'axe Y = (324,678)

les valeurs à associer aux extrémités-origine en sont déduites.

figure 4.13

Nous appelerons fourchette d'un axe l'ensemble des valeurs comprises entre la valeur associée à l'extrémité-origine et la valeur associée à l'extrémité finale de l'axe. La fourchette est toutefois définie par l'utilisateur en précisant les valeurs associées à l'intersection des axes (ou aux extrémités-origine des axes si ceux-ci ne se recoupent pas) et aux extrémités finales des axes.

Ce choix a été fait dans le but d'éviter à l'utilisateur des calculs inutiles s'il désire qu'une valeur précise soit affectée à l'intersection des axes.

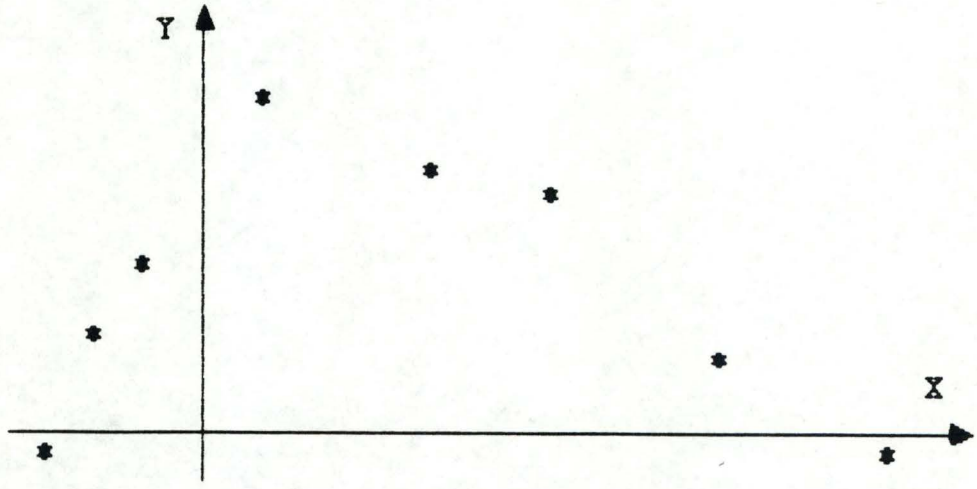
La valeur à affecter à l'extrémité-origine d'un axe peut facilement être déduite (calculée) à partir des propriétés de l'axe (coordonnées de l'extrémité-origine et longueur de l'axe) et des valeurs déterminant la fourchette (valeur affectée à l'intersection des axes et de l'extrémité finale).

Une option par défaut est prévue. Celle-ci dispense l'utilisateur de déterminer lui-même la fourchette. Dans cette option, les valeurs associées aux extrémités de l'axe des abscisses sont les minimum et maximum de l'ensemble des valeurs d'abscisse des relations à illustrer sur un même graphe. De même, les valeurs à affecter aux extrémités de l'axe Y sont les valeurs extrêmes de l'ensemble des valeurs d'ordonnée de ces relations.

Dans l'option par défaut, tous les points des relations à illustrer par rapport à un même système d'axes tombent dans la fourchette (voir figure 4.14).

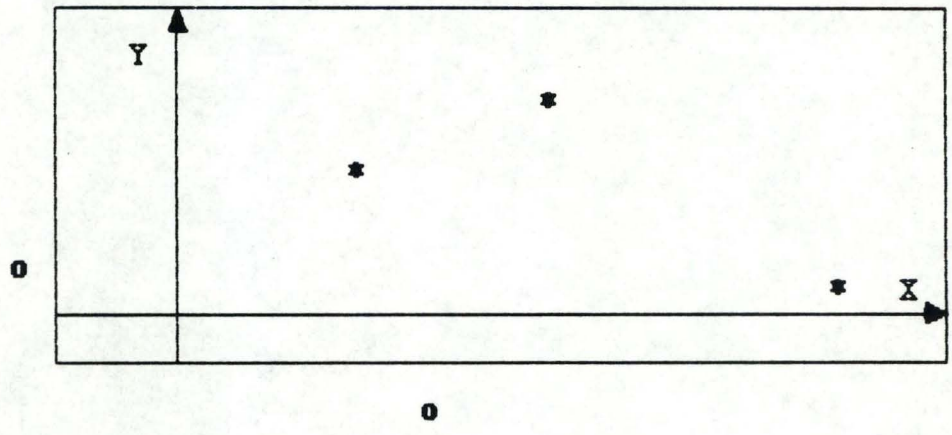
Une telle affirmation n'est plus nécessairement vérifiée dans le cas où l'utilisateur détermine lui-même la fourchette. Dans ce cas, les points dont les projections ne tombent pas dans les fourchettes définies sur les axes ne sont pas imprimés (voir figure 4.15).

Remarquons que si la fourchette est déterminée par l'utilisateur et que les points d'une relation sont reliés entre eux , des situations critiques peuvent se présenter. Si un point n'est pas imprimé parce que ses coordonnées ne tombent pas dans les fourchettes , les points qui lui sont voisins ne peuvent être reliés entre eux. Il faut donc tenir compte , pour tracer les liens entre les points , des points tombant en dehors des fourchettes (voir figure 4.16).



Dans l'option par défaut, tous les points de la relation tombent dans les fourchettes.

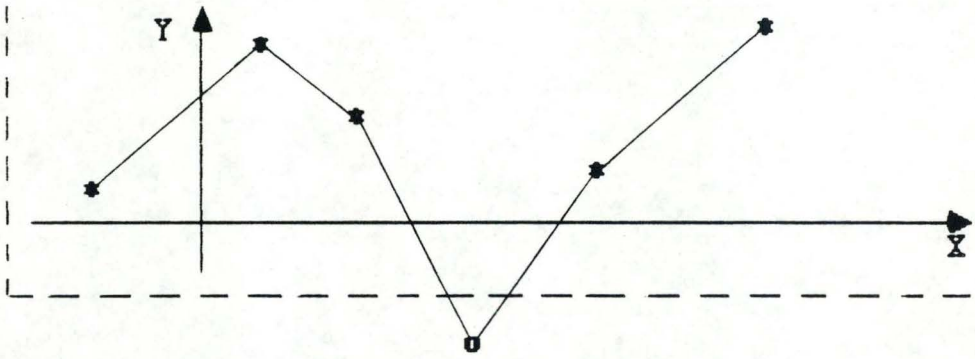
figure 4.14



= zone d'affichage

Les points dont les projections ne tombent pas dans les fourchettes ne sont pas imprimés (o).

figure 4.15



o : point tombant en dehors de la zone d'affichage

proposition d'affichage des liens :

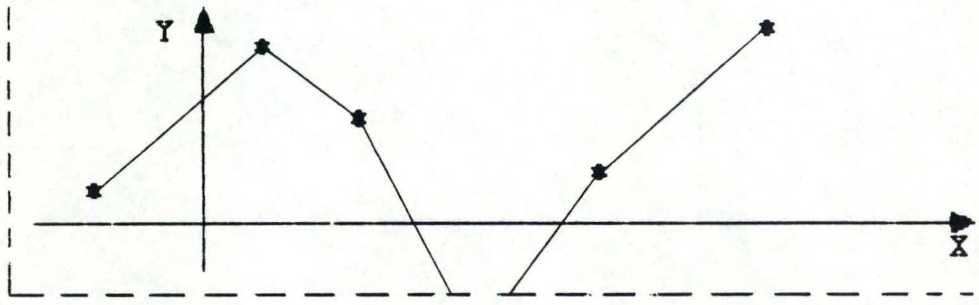


figure 4.16

4.8 Coordonnées objets et coordonnées sources d'un point

Tout point d'une courbe est imprimé à un endroit précis de la zone d'affichage déterminé par ses coordonnées (abscisse et ordonnée), et par la fourchette associée à chacun des axes.

Nous appelerons coordonnées objets d'un point les distances, dans une certaine unité, des projections de ce point à l'intersection des axes, et coordonnées sources les valeurs que ce point représentent dans le graphe (valeurs contenues dans le tableau de résultats).

Il existe, de façon générale, une certaine relation entre les valeurs des coordonnées objets et des coordonnées sources d'un même point. Cette relation dépend des valeurs associées aux extrémités de chaque axe. Elles dépend donc des fourchette des axes.

4.9 Echelle d'un axe

La relation entre les deux types de coordonnées peut appartenir à différents types que nous appelerons échelles.

Si l'échelle est linéaire , la relation entre les deux types de coordonnées (source et objets) est linéaire. Cette relation est de la forme : $o = a.s + b$, où o représente la coordonnée objet et s la coordonnée source. Les paramètres a et b sont entièrement déterminés quand on connaît la fourchette.

En pratique, outre les échelles linéaires, sont souvent utilisées les échelles logarithmiques. Si l'échelle est logarithmique , la relation entre les deux coordonnées est de la forme : $o = a \log_c s + b$, o et s étant respectivement la coordonnée objet et la coordonnée source. Les paramètres de cette relation sont entièrement déterminés si , outre la fourchette , on précise la base du logarithme (c).

Il est nécessaire d'isoler dans un module facile d'accès tout ce qui, lors de l'illustration d'une relation dans un système d'axes, dépend de l'échelle utilisée. Ce module est un interface entre les coordonnées sources d'un point et ses coordonnées objets. Ceci permet de ne pas se limiter aux échelles linéaire et logarithmique, mais d'offrir la possibilité à l'utilisateur de prévoir tout autre type d'échelle à condition qu'il écrive le module d'interfaçage correspondant.

4.10 Etalonnage et marques

Nous appelerons étalonnage une série de repères imprimés sur un axe et accompagnés de l'impression de la valeur source correspondante (voir figure 4.17).

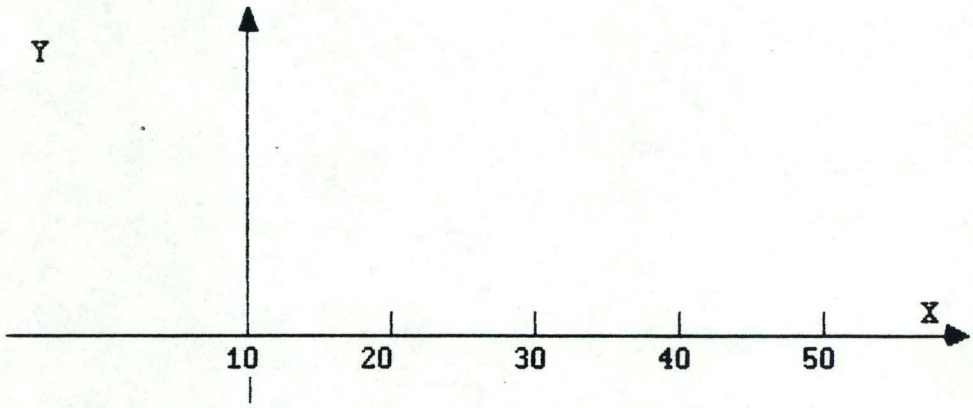
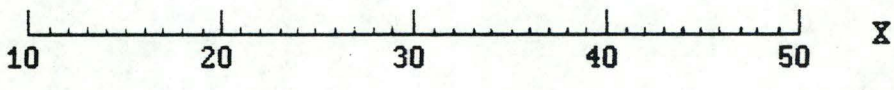


Illustration d'un étalonnage sur l'axe des X pour une échelle linéaire.

figure 4.17

Nous appelerons marque un repère imprimé sur un axe mais non accompagnée de l'impression d'une valeur .

La fréquence d'impression d'une marque est en général plus élevée que celle d'un étalonnage sur le même axe, ce qui a pour effet qu'entre deux repères d'étalonnage consécutifs sont imprimées une ou plusieurs marques (voir figure 4.18).



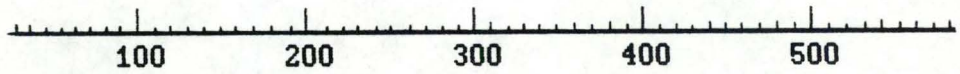
Etalonnage et marquage combinés

figure 4.18

L'impression des valeurs accompagnant les repères d'étalonnage se fera entièrement à condition qu'il y ait suffisamment de place pour les imprimer toutes, et qu'il n'y ait donc pas de chevauchement de caractères. S'il y a chevauchement, un nombre sur deux sera imprimé. Si il y a toujours chevauchement, la fréquence d'impression des valeurs sera encore diminuée.

La fréquence d'impression des repères d'étalonnage et des marques est analysée ci-dessous pour les types d'échelle linéaire et logarithmique. L'impression des marques et étalonnages relatifs à tout autre type d'échelle devra être gérée dans le module d'interfaçage écrit par l'utilisateur désireux de travailler avec une autre échelle.

Dans le cas de l'échelle linéaire, la différence entre les valeurs extrêmes de la fourchette est calculée, et exprimée en puissance maximum de 10. Par exemple, le nombre 768,5 sera transformé en $7,685 . 10^2$. La puissance de 10 ainsi obtenue (soit 10^x) indique la fréquence d'impression des repères. Toutes les 10^x unités, un repère d'étalonnage et la valeur correspondante sont affichés, et toutes les 10^{x-1} unités, une marque sera imprimée (voir figure 4.19).



Bornes de la fourchette : 13.5 et 582
différence = 568.5 ou 5.685 * 10^2
fréquence des repères d'étalonnage : 10^2
fréquence des marques : 10^1

figure 4.19

L'utilisateur peut choisir d'augmenter cette fréquence, en modifiant ce que nous appelerons le degré de résolution. Le degré de résolution par défaut est 1. Si l'utilisateur le choisi égal à p, la fréquence d'impression des repères est multipliée par p. Les repères d'étalonnage seront imprimés toutes les 10^x / p unités, et les marques toutes les 10^{x-1} / p unités.

Dans le cas de l'échelle logarithmique, un repère d'étalonnage est affiché aux endroits correspondant aux valeurs entières du $\log_c s$ (s = coordonnée source et c = la base du logarithme).

Dans l'exemple où la fourchette est comprise entre 1 et 10 000 et le logarithme est en base 10, cinq repères sont imprimés dont les valeurs sont : 1, 10, 100, 1000 et 10 000 (voir figure 4.20). Si la base est 4 et la fourchette comprise entre 2 et 68, trois repères d'étalonnage sont imprimés dont les valeurs sont 4, 16 et 64 (voir figure 4.21).

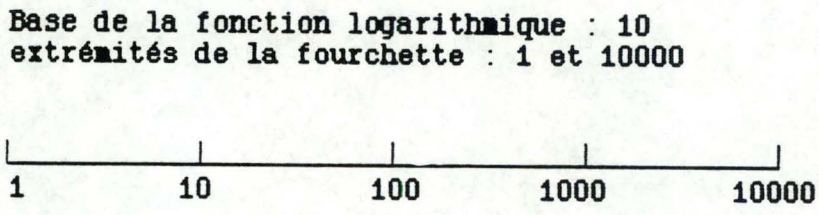


figure 4.20

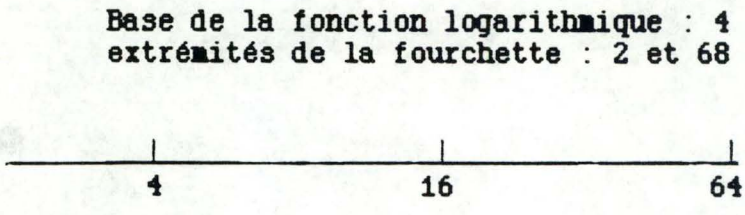


figure 4.21

En général, si le logarithme est en base c, les marques, entre deux repères consécutifs d'étalonnage dont les valeurs sont c^x et c^{x+1} , sont imprimées aux endroits correspondant à toutes les c^x unités. Dans l'exemple précédent, entre les repères d'étalonnage accompagnés des valeurs 10^2 et 10^3 , les marques sont imprimées aux endroits correspondant à toutes les 10^2 unités, c'est-à-dire 200, 300, 400, ... 900 (voir figure 4.22).

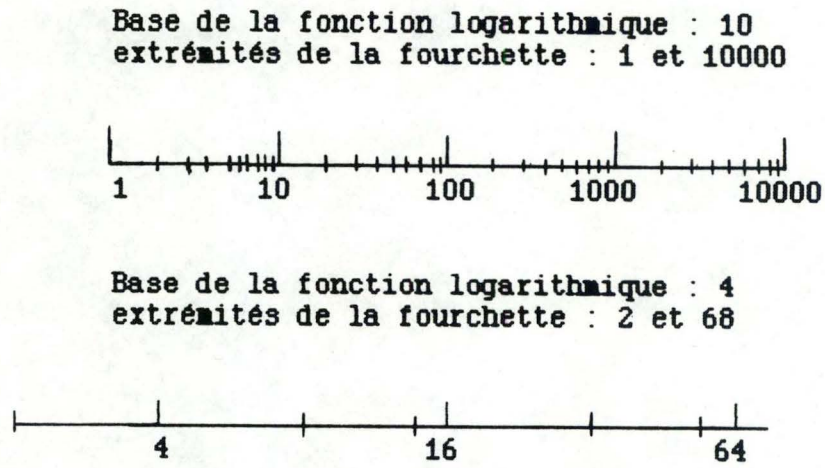


figure 4.22

Remarquons que si l'on veut faire varier la fréquence d'étalonnage dans le cas de l'échelle logarithmique , on peut jouer sur la base du logarithme.

4.11 Classe et intervalle de classe

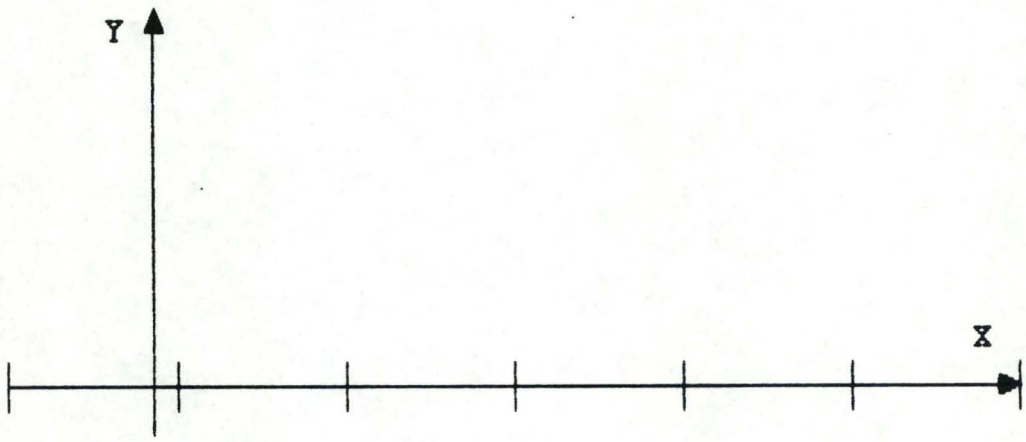
L'ensemble des abscisses des points d'une relation peut être divisé en une série de classes contigüs. Nous appellerons classe une série de valeurs délimitée par une borne inférieure et une borne supérieure, et contenant les valeurs d'abscisses d'une série de points consécutifs d'une relation. La borne inférieure fait partie de la classe, la borne supérieure appartient à la classe suivante.

Nous appelerons intervalle de classe l'intervalle délimité par la borne supérieure et la borne inférieure d'une classe.

Les intervalles de classes peuvent être définis par l'utilisateur. La découpe en classes peut être déterminée soit par le nombre de classes total désiré, soit par la largeur de l'intervalle de classe (voir figures 4.23 et 4.24). Dans chacune de ces alternatives, la largeur de l'intervalle des classes est constant, à l'exception éventuellement de la dernière classe. En effet, il est possible que l'utilisateur choisisse un intervalle de classe ayant pour conséquence un nombre de classes non entier. Dans ce cas, la dernière classe sera moins large que les autres.

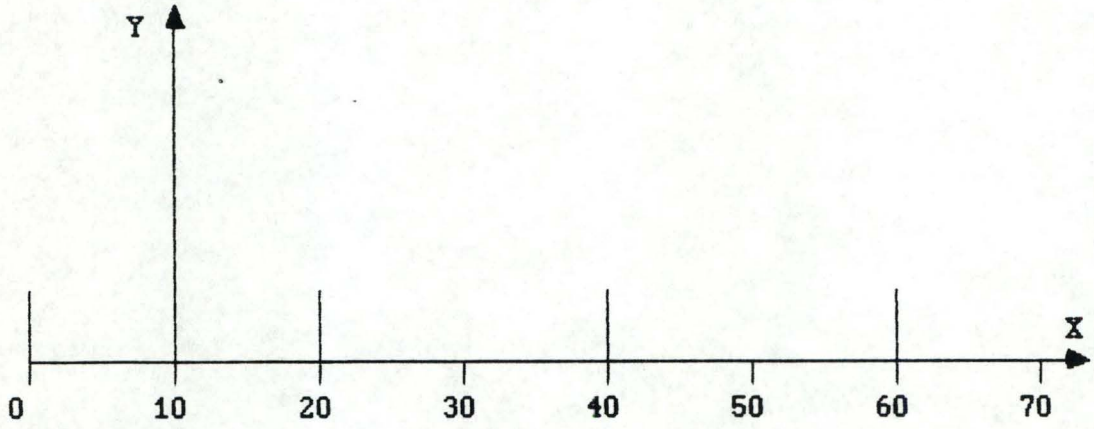
On pourrait étendre les mécanismes de détermination des classes. On pourrait offrir à l'utilisateur la possibilité de définir explicitement chaque limite de classe à l'aide d'un vecteur.

L'intervalle de classe pourrait même être défini en fonction des valeurs (d'une variable) reprises dans le tableaux de résultats. L'intervalle d'une classe pourrait, par exemple, être proportionnel (ou inversément proportionnel) à la moyenne des ordonnées des points dont les abcisses tombent dans cette classe (voir figure 4.25).



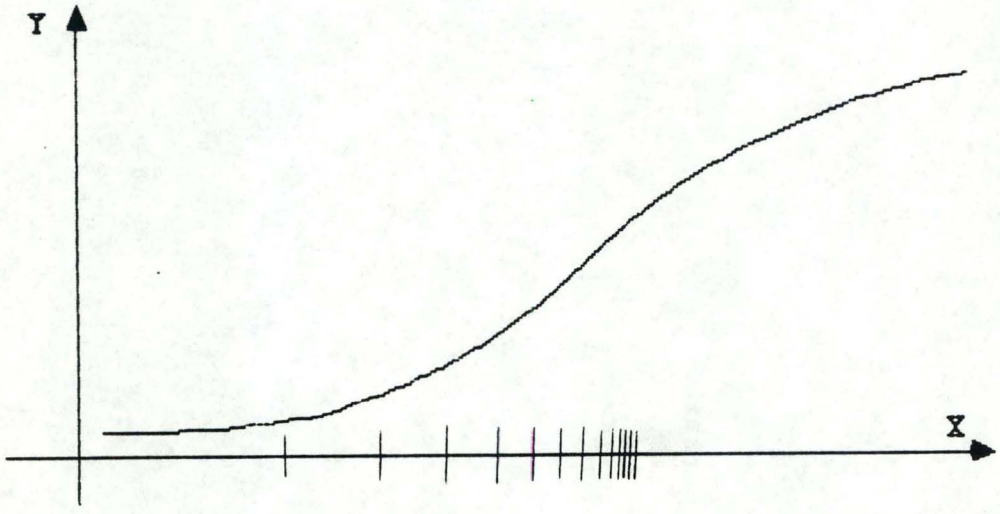
Soit un nombre de classes donné = 6
Les intervalles sont constants lorsqu'on donne un nombre de classes.

figure 4.23



soit un intervalle de classe de 20

figure 4.24



Intervalle de classe inversément proportionnel à la moyenne des ordonnées des points dont les abscisses tombent dans la classe.

figure 4.25

4.12 Diagramme en bâtonnets

L'ensemble des abscisses des points de la relation observée est découpé en un certain nombre de classes; chaque classe est délimitée par une borne supérieure et une borne inférieure. La moyenne des ordonnées des points dont l'abscisse tombe dans une même classe est calculée, et détermine la hauteur de la colonne correspondante à cette classe (voir figure 4.26). Nous appelerons diagramme en batônnets d'une relation l'ensemble des colonnes ainsi construites.

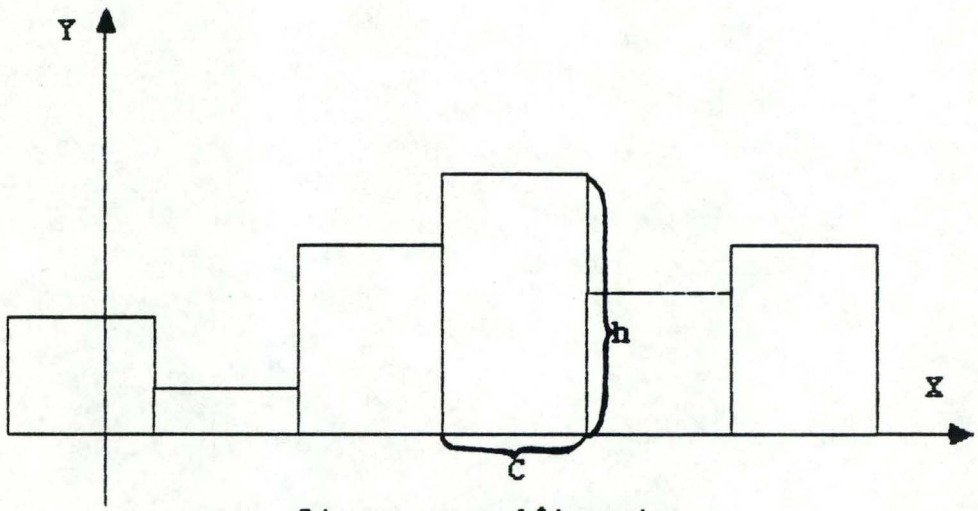


Diagramme en bâtonnets

La hauteur (h) de la colonne est obtenue par le calcul de la moyenne des ordonnées des points dont l'abscisse tombe dans l'intervalle de classe C.

figure 4.26

Un tel diagramme est donc une représentation graphique particulière d'une relation entre deux variables.

Un diagramme en bâtonnets est déterminé par les coordonnées d'une relation un tracé et les intervalles de classe. Un tel diagramme est tracé par rapport à un système d'axes (voir concept d'axes, de fourchette et d'échelle).

4.13 Histogramme

Un histogramme est une manière particulière de visualiser les valeurs reprises dans une colonne d'une matrice d'un tableau de résultats. Un histogramme reprend comme abscisses les valeurs d'une colonne de matrice, et comme ordonnées la fréquence de ces valeurs dans cette colonne de la matrice. Ce genre de diagramme étant fréquent en biologie, il sera donc prévu dans le cadre de ce progiciel (voir figure 4.27).

Il ne s'agit donc pas de l'illustration d'une relation entre deux variables d'un tableau de résultats. Un histogramme est déterminé si l'on précise le tableau de résultats, la variable à illustrer et les intervalles de classe.

Soit la suite de valeurs:

7, 3, -2, -1, 7, -1, -1, 3, 3, -1, 4, 5, 3, 0, 4

soit un intervalle de classe de 2

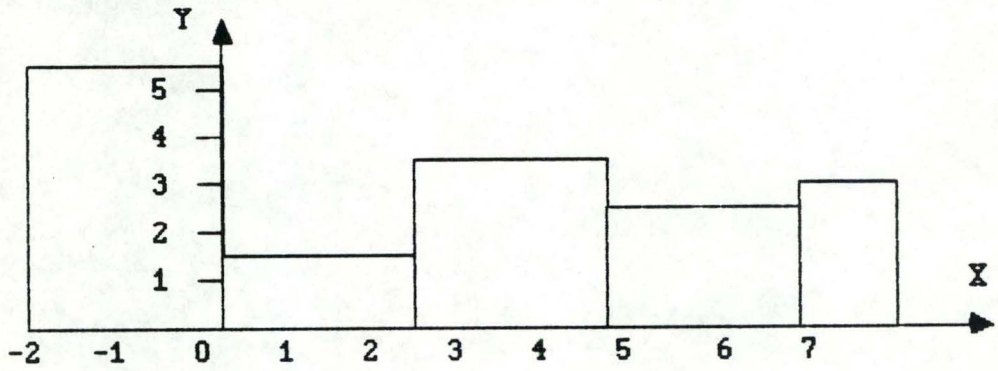


figure 4.27

L'axe des abscisses relatif à un histogramme ne diffère en rien de la notion d'axe utilisée jusqu'ici. Une fourchette et une échelle y sont asssociées. L'axe des ordonnées, quant à lui, est différent. Puisqu'il reprend des valeurs de fréquence, sa fourchette sera définie par défaut. L'intersection des axes aura la valeur 0, et l'extrémité finale prendra la valeur de la fréquence la plus élevée. La portion éventuelle de droite en dessous de l'intersection des axes est non significative.

Une option peut être prévue. Dans celle-ci, l'axe des ordonnées représentera la fréquence relative. L'extrémité finale de l'axe Y prendra donc la valeur 1.

Le format du tracé des colonnes des diagrammes en batonnets et histogrammes est choisi par l'utilisateur selon un code. Ce concept est donc différent du concept de patron de tracé défini pour les courbes. Mais les remarques concernant le code des formats des liens d'une courbe (voir concept de patron de tracé d'une courbe) sont également valables quant aux modifications possibles de ce code par l'utilisateur.

Plusieurs diagrammes ou histogrammes peuvent être tracés par rapport au même système d'axes. Ces différents diagrammes peuvent être facilement différenciés par l'utilisateur final, car la possibilité est offerte de définir un tracé différent pour chaque diagramme.

4.14 Graphe

Un graphe est un ensemble de courbes, diagrammes en bâtonnets ou histogrammes tracés par rapport à un même système d'axes. Sur chaque axe sont définies une échelle et une fourchette. Le système d'axes est lui-même tracé à l'intérieur d'une fenêtre. A un graphe est associé un seul titre de graphe et un seul système d'axes. Par conséquent, toutes les relations d'un graphe seront tracées par rapport au même axe X et même axe Y. Ceci implique qu'un seul titre d'axe et qu'une seule unité seront affectés à chaque axe.

Les fourchettes de chacun des axes seront déterminées, dans l'option par défaut, à partir des points de l'ensemble des relations à illustrer sur le même graphe. Sur un même graphe peuvent être illustrées des relations à la fois sous forme de courbes et de diagramme en bâtonnets.

Puisque plusieurs relations tracées sur un même graphe le sont par rapport au même système d'axes, la fourchette de l'axe des abscisses (des ordonnées) sera déterminée à partir de l'ensemble des valeurs d'abscisse (d'ordonnées) des différentes relations.

La deuxième option prévoit de mettre à jour les fourchettes (si cela est nécessaire) de sorte que tous les points des relations à visualiser sur un même graphe puissent être imprimés. Si les fourchettes sont modifiées, les relations déjà imprimées doivent être retraçées pour être cohérentes avec les nouvelles fourchettes. Cette option impliquerait donc la mémorisation de la constitution d'un graphe. Ceci introduit la notion d'existence logique d'un graphe. Il faut , en effet , mémoriser certains des paramètres qui déterminent le graphe , tels que le type d'échelle ou les caractéristiques des différentes relations illustrées sur le graphe. Pour des raisons d'efficacité et de précision , il est préférable que les tableaux de résultats reprenant les valeurs des relations soient toujours disponibles.

4.15 Amélioration

Jusqu'ici, nous n'avons envisagé que des axes orientés dans un seul sens: vers la droite pour l'axe X, vers le haut pour l'axe Y.

L'illustration de certaines applications sous forme graphique sera peut-être plus explicite s'il est possible d'orienter les axes différemment. Ce peut être le cas de l'illustration de la densité d'une population marine en fonction de la profondeur océanique. Si la profondeur est représentée en ordonnée, il est intéressant d'orienter l'axe Y vers le bas.

Cette possibilité peut par conséquent être envisagée dans le cadre de ce travail.

D'autre part, nous n'avons jamais fait allusion qu'à un seul axe Y dans un système d'axes. Il arrive cependant qu'une même variable en conditionne deux autres, sans pour autant que ces dernières partagent la même unité. La difficulté de représenter alors sur un même graphe ces deux relations peut être contournée si l'on prévoit la possibilité de tracer deux axes Y dans une même fenêtre. Dans un tel système d'axes, chaque relation tracée se rapporte à un des deux axes Y.

Les graphes tridimensionnels n'ont pas non plus été abordés jusqu'ici. Ce type de graphe peut parfois être moins lisible qu'un graphe à deux dimensions, mais néanmoins être utile dans certaines applications. Ils peuvent par conséquent être proposés pour étendre les possibilités offertes par le système.

Il existe une manière souvent plus intéressante de représenter une relation entre trois variables. C'est notamment le cas lorsqu'on veut illustrer deux variables (A et B) en fonction d'une troisième (par exemple le temps (C)). Un système de deux axes est tracé, et la relation entre A et B est illustrée sous forme d'une courbe. Chaque point de la courbe est accompagné de la valeur correspondante de C. Toute valeur (de C) à imprimer en un point de la courbe se trouve dans la matrice du tableau de résultats sur la même ligne que les valeurs des deux autres variables (A et B) identifiant ce point (voir figure 4.28). Cette option peut aussi être prévue dans le système.

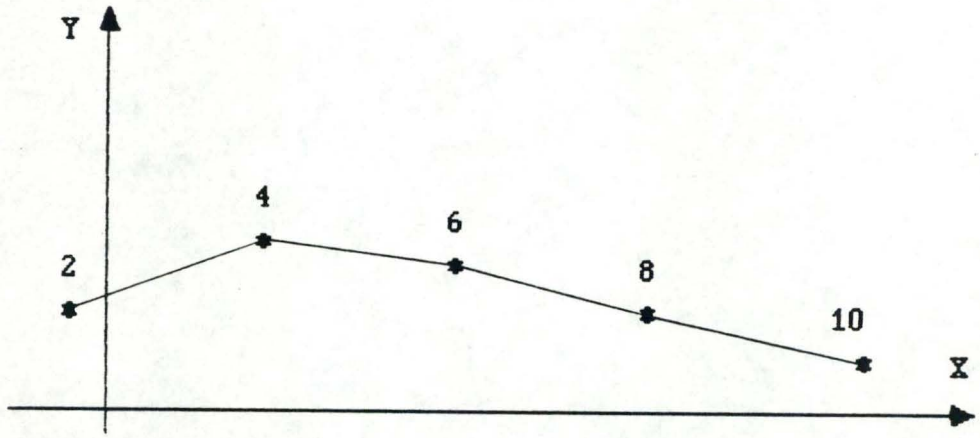


figure 4.28

CHAPITRE 5 : AUTRES CONCEPTS

Nous proposons ci-dessous une série d'outils qu'il paraît intéressant d'intégrer au progiciel. Ces outils et les concepts qui y sont associés ne font pas ici l'objet d'une analyse détaillée. Il s'agit surtout de propositions.

5.1 Affichage des valeurs d'un tableau de résultats

Nous avons déjà analysé les concepts relatifs à une première forme de visualisation des résultats repris dans un tableau de résultats, à savoir le graphe. Ces concepts ont été développés dans la section précédente.

Nous proposons ici une autre manière de visualiser les résultats. En effet, bien que l'illustration graphique des résultats soit fort utilisée dans l'enseignement, il peut être intéressant de communiquer à l'élève les valeurs des résultats de simulation avec plus de précision.

Nous désirons par conséquent offrir à l'utilisateur final la possibilité d'avoir accès aux valeurs contenues dans la matrice d'un tableau de résultats donné. Nous pourrions appeler cet outil affichage d'une matrice dont les principes de base seraient les suivants.

Les valeurs se trouvant sur une même ligne dans la matrice sont affichées sur une même ligne à l'écran. De même, les valeurs appartenant à une même colonne sont imprimées sur un même colonne.

Il est évident que, dans beaucoup de cas, il est impossible d'afficher sur un même écran l'ensemble des valeurs d'une matrice en une fois. L'élève n'a alors accès qu'à une partie de la matrice. Les limites de l'écran pourraient être considérées comme les limites d'un cadre ou d'une fenêtre à travers laquelle l'élève pourrait visualiser une partie de la matrice.

S'il est impossible d'afficher l'entièreté de la matrice à l'écran, seulement une partie des lignes et une partie des colonnes de la matrice sont accessibles, c'est-à-dire visualisables à travers la fenêtre que délimite l'écran.

Il est alors nécessaire de fournir à l'élève la possibilité de "déplacer" cette fenêtre à travers la matrice, ce qui lui permettrait d'avoir accès à l'ensemble de la matrice. Ce déplacement pourrait se faire dans quatre directions: vers le haut, vers le bas, vers la droite et vers la gauche. A chaque déplacement de la fenêtre, une autre partie de la matrice est accessible.

Nous proposons d'accompagner l'affichage d'une (partie de) ligne de l'impression du numéro d'ordre de cette ligne dans la matrice. De même, chaque colonne peut être accompagnée du nom (et éventuellement de l'unité) de la variable à laquelle elle correspond dans le tableau de résultats. Ceci est proposé dans le but d'aider l'élève à se situer dans la matrice. L'impression du numéro de ligne pourrait par exemple se faire à la gauche de chaque ligne, et le nom (et l'unité) de la variable au dessus de chaque colonne.

On peut s'inspirer aussi de certaines possibilités offertes par les tableurs.

5.2 Gestion des entrées-sorties

Le dialogue avec l'élève tient une place importante dans beaucoup de didacticiels. Dans le cas de simulations, ce dialogue peut avoir pour effet de faire participer l'utilisateur final à la détermination de l'expérience à simuler.

Dans bien des cas, le constructeur du didacticiel doit gérer des entrées et des sorties, et notamment analyser les données introduites par l'élève pour s'assurer de leur cohérence. Si, par exemple, la donnée à introduire doit être de type entier, il n'est pas certain que l'élève n'introduira pas de caractères autres que numériques. Chaque donnée introduite doit donc faire l'objet de vérifications.

Nous désirons soulager le constructeur du didacticiel de ce genre de tâche. C'est pourquoi nous proposons ci-dessous un certain nombre d'outils qui s'assurent que la donnée introduite par l'élève répond à certaines contraintes.

Les outils qu'il nous semble utile de prévoir sont:

- lecture (à partir du clavier) d'une suite de caractères de longueur maximale donnée. Chaque caractère introduit doit faire partie d'un ensemble de caractères (dits acceptables). Si un caractère n'est pas acceptable, il n'est pas affiché à l'écran, ni inscrit dans la suite de caractères constituant la donnée.

- lecture d'un entier signé ou non

- lecture d'un réel (différents formats pourraient être envisagés, comme c'est d'ailleurs généralement le cas en programmation. Ex: 700.43 = 7.0043E2)

- lecture d'un entier ou réel satisfaisant une contrainte particulière, par exemple: . inférieur à une valeur donnée
  . supérieur à une valeur donnée
  . compris entre deux bornes données

Cette liste n'est évidemment pas exhaustive.

Les outils proposés ci-dessus sont relativement simples. On pourrait également en concevoir de plus élaborés en s'inspirant par exemple des gestionnaires d'écran existants.

5.3 Génération de nombre aléatoire

Nous avons souvent fait allusion à des simulateurs basés sur des calculateurs qui utilisent un générateur de nombres aléatoires pour calculer les valeurs des variables dépendantes.

Dans certaines expériences biologiques, la dimension aléatoire est importante. On peut citer comme exemple des échantillons prélevés aléatoirement dans le temps à un même endroit. Les échantillons peuvent dans d'autres cas être prélevés à divers endroits choisis de manière aléatoire sur un territoire donné.

D'autre part, des outils de génération de nombres aléatoires peuvent être utiles dans le cas où l'on désire simuler la variabilité expérimentale à partir de résultats déterministes. On peut simuler la variabilité expérimentale à partir d'une valeur donnée, en générant un nombre suivant une distribution (normale par exemple) de moyenne égale à ce nombre.

CHAPITRE 6 : CHOIX D'IMPLEMENTATION

Nous avons implémenté un sous-système utile qui est constitué de :

- une gestion des matrices faisant partie des tableaux de résultats
- l'implémentation du simulateur
- certains outils de création et manipulation de tableaux de résultats
- quelques primitives d'entrée-sortie dont un outil de visualisation des résultats repris dans une matrice.

6.1 Les tableaux de résultats

6.1.1 Discussion

Nous nous trouvons devant un choix important à faire à propos de l'implémentation des tableaux de résultats

Rappelons qu'un tableau de résultats est constitué d'une part d'un vecteur-définition et d'autre part d'une matrice. Le vecteur-définition reprend pour chaque variable du tableau de résultats sa définition. La matrice, quant à elle est un tableau à deux dimensions; les valeurs reprises dans la ième colonne de la matrice sont associées à la variable identifiée par la ième définition du vecteur-définition. La notion d'ordre est donc importante tant au niveau des définitions dans le vecteur-définition qu'au niveau des lignes et colonnes dans la matrice.

L'implémentation du vecteur-définition sous forme d'un tableau à une dimension (vecteur) de records en Pascal, et de la matrice sous forme d'un tableau à deux dimensions de réels semble à première vue évidente. Cependant, en Turbo-Pascal comme en Pascal, les tableaux à bornes dynamiques n'existent pas. Ceci signifie que lors de la déclaration des tableaux, les bornes doivent être fixées et donc connues. Or, il n'est pas toujours possible de connaître, avant l'exécution du didacticiel, la valeur des bornes des tableaux. C'est entre autre le cas lorsque la participation de l'élève est requise pour déterminer l'expérience à simuler.

Les outils intégrés à notre système travaillent sur des tableaux dont les matrices sont de bornes variables. Or, ils doivent pouvoir être écrits indépendamment des dimensions des tableaux qu'ils manipulent.

D'autre part, les vecteurs-définition contiennent autant d'éléments qu'il y a de variables à définir. Quant aux matrices, si elles contienent autant de colonnes qu'il y a de variables, elles contiennent aussi autant de lignes qu'il y a d'observations à simuler. Le nombre d'observations simulées peut devenir important et, contrairement au nombre des variables, dépasser facilement plusieurs dizaines.

Le nombre de valeurs à mémoriser lors d'une simulation peut devenir important. Le nombre de tableaux de résultats manipulés lors de l'exécution du didacticiel peut, en outre, ne pas être négligeable. Ceci augmente encore le nombre d'éléments à stocker. La mémorisation des résultats des simulations peut donc mobiliser de la place mémoire en quantité non négligeable. Il est dès lors indispensable d'optimiser l'utilisation de la place mémoire.

Ces diverses observations montrent que, d'une manière générale, il est intéressant de simuler des tableaux à bornes dynamiques. Dans cet ordre d'idée, trois cas sont envisageables.

a. Premier cas :

Une première possibilité, mais de loin la moins efficace, est de considérer qu'il n'existe qu'une seule sorte de vecteur et de matrice ayant tous deux une capacité maximale permise. Cette capacité maximale doit être suffisante pour contenir les résultats de simulation d'expériences portant sur un nombre élevé de variables et composées d'une suite considérable d'observations. Tous les vecteurs-définition sont donc de même longueur, même si uniquement certains éléments du vecteur sont utilisés. De même, les matrices faisant partie de n'importe quel tableau de résultats contiennent le même nombre de lignes et le même nombre de colonnes, alors que seulement les valeurs reprises dans certaines lignes et colonnes de la matrice sont significatives.

En d'autres termes, la longueur physique d'un vecteur-définition peut différer de sa longueur logique. De même, le nombre logique de lignes et de colonnes d'une matrice peut différer du nombre physique.

Ceci représente un gaspillage évident de place mémoire, puisque la plus petite des expériences simulées (peu de variables et peu d'observations) réservera d'office un vecteur-définition et une matrice de capacité maximale.

b. Deuxième cas :

Une autre possibilité consiste à travailler avec des lignes de matrices et des vecteurs-définition en longueur fixe, tandis que le nombre de lignes est variable d'une matrice à l'autre . Dans cette optique, la longueur logique d'un vecteur-définition peut différer de sa longueur physique. Il en est de même pour le nombre de colonnes d'une matrice. Il n'y a plus de gaspillage de lignes, ce qui entraîne un gain de place. on gaspille malgré tout des colonnes. Cependant, cet inconvénient est moindre que si l'on avait considéré l'implémentation symétrique dans laquelle le nombre de colonnes serait variable et le nombre de lignes constant. Ceci s'explique par le fait que la variabilité du nombre de colonnes (c'est-à-dire de variables) est moindre que la variabilité du nombre de lignes (c'est-à-dire d'observations à simuler), tout au moins au sein d'un didacticiel.

c. Troisième cas :

Une troisième possibilité qui éviterait tout gaspillage de place mémoire est de simuler des vecteurs-définitions et des matrices à bornes dynamiques. Il n'y aurait plus alors à faire de distinction entre les dimensions physiques et logiques des entités en jeu.

La première implémentation est impraticable car elle conduit à des limitations trop importantes sur la taille et le nombre de tableaux de résultats utilisables. La dernière implémentation quant à elle est complexe.

Etant donné que l'implémentation que nous proposons constitue seulement un prototype du système, nous avons opté pour la deuxième solution. Celle-ci est moins efficace du point de vue place mémoire mais est plus simple à réaliser. Si nécessaire, il sera possible, ultérieurement, de modifier l'implémentation des tableaux de résultats moyennant la réécriture des primitives d'accès correspondantes. Ces primitives sont marquées ♦♦♦ en annexes (voir annexe 1).

6.1.2 Implémentation

a. Vecteur-définition

Les vecteurs-définition sont implémentés sous forme de tableaux à une dimension de longueur égale à une constante appelée 'largmax'. Cette constante est déclarée en tête du texte du progiciel, et peut être facilement modifiée. Celle-ci est arbitrairement fixée à 20 dans notre prototype.

```
const   largmax = 20
```

Chaque élément du vecteur-définition est de la forme:

```
type : variable = record nom : strnm;
                         unité : strut;
                         case genre : genrevar of
                         vd : ( );
                         vi : (case md : modet of
                              con : ( );
                              nonc : ( ) );
                         ve : (origine:real) end;
```

Il s'agit de la définition d'une variable du simulateur. Le nom est une chaîne de caractères de longueur égale à 8 (strnm) et l'unité est une chaîne de caractères de longueur égale à 10 (strut).

```
type  strnm : string[8];
      strut : string[10];
```

Si l'on désire modifier la longueur du nom ou de l'unité, il suffit de modifier la déclaration de strnm ou strut, placés au début du texte du programme.

Les types 'genrevar' et 'modet' sont déclarés comme suit:

```
type  genrevar = (vi, vd, ve);
      modet = (con, nonc);
```

'genrevar' permet de spécifier le genre des variables : indépendante (vi), dépendante (vd) et évolutive (ve);
'modet' précise, s'il s'agit d'une variable indépendante, le mode de détermination : connue a priori (con) ou non (nonc).

Une variable indépendante ne requiert aucune information supplémentaire (vd:( )). La seule information accompagnant une variable évolutive est sa valeur avant le début de l'expérience (origine).

Dans le cas d'une variable indépendante, si elle est connue a priori (md = con) aucune information supplémentaire n'est nécessaire. Tandis que si elle n'est pas connue a priori, plusieurs renseignements sont requis pour calculer sa valeur à chaque observation. Le calcul de telles valeurs n'a pas été implémenté mais a été prévu dans la structure de la définition d'une variable. Lors de l'implémentation de ce mode de détermination, il faudra compléter l'option (nonc: ()). Au niveau du simulateur, cette option a également été prévue (voir implémentation du simulateur).

Tout vecteur-définition est de la forme :

```
vect : array [1..largmax] of variables
```

L'ordre logique des définitions correspond ici à l'ordre physique des éléments dans le tableau à une dimension.

Il est important de noter que toute autre implémentation du vecteur-définition est permise à condition que la structure de la définition d'une variable ne soit pas modifiée, si ce n'est pour ajouter les informations relatives aux variables indépendantes non connues a priori.

De plus, quelle que soit l'implémentation des vecteurs-définition et des matrices, une valeur devra être affectée à 'largmax'. Cette valeur représente le nombre maximum de variables qui peuvent être associées à un tableau de résultats. Cette constante est en fait utilisée à plusieurs reprises pour définir des tableaux à une dimension nécessaires à l'implémentation du simulateur.

b. Matrice

Comme nous l'avons dit précédemment, nous avons choisi d'optimiser l'utilisation de la place mémoire en simulant des tableaux à deux dimensions à borne dynamique uniquement en ce qui concerne le nombre de lignes. Le nombre physique de colonnes de n'importe lequel de ces tableaux est constant, même si toutes les colonnes ne sont pas utilisées.

Toute ligne de matrice est un tableau à une dimension de longueur égale à 'largmax'. Tout élément d'une matrice est un réel.

Nous avons implémenté les matrices de la façon suivante.

Les lignes de n'importe quelle matrice font toutes partie d'un seul et unique tableau à deux dimensions dont le nombre de colonnes est égal à 'largmax' et le nombre de lignes est égal à une constante appelée 'longmax'. Comme 'largmax', cette constante est déclarée en tête du texte du progiciel, ce qui permet de la modifier facilement. Celle-ci est fixée arbitrairement à 500 dans ce prototype.

```
const longmax = 500
```

Ce tableau à deux dimensions est déclaré comme suit :

```
tabval : array [1..longmax, 1..largmax] of real;
```

Si une matrice de i lignes doit être attribuée à un tableau de résultats donné, i lignes consécutives sont réservées dans le tableau 'tabval'. Le numéro de la première ligne réservée dans tabval est transmis au tableau de résultats et y est mémorisé, de même que le nombre de lignes réservées. La réservation de lignes consécutives dans 'tabval' est possible uniquement s'il y a ou s'il reste suffisamment de lignes libres. Il faut donc avoir prévu un tableau ('tabval') suffisamment grand, c'est-à-dire contenant suffisamment de lignes.

Une manière d'économiser les lignes est de libérer les lignes réservées lors de l'attribution d'une matrice dès que cette matrice n'est plus exploitée dans la suite (de l'exécution) du didacticiel.

Il faut, par conséquent, mémoriser les lignes dans 'tabval' qui sont occupées (réservées) ou libres. La libération, ou la réservation, de lignes dans 'tabval' oblige une optimisation de la gestion des lignes de 'tabval'. Le fait que les lignes réservées par un tableau de résultats soient consécutives demande parfois le regroupement des lignes libres mais disséminées dans 'tabval' (donc non consécutives). Ceci implique le déplacement (recopie) des lignes constituant la matrice de certains tableaux de résultats, mais aussi la mise à jour du numéro de la première ligne réservée dans 'tabval' et mémorisée dans les tableaux de résultats dont les matrices ont été "déplacées" dans 'tabval'.

Ici également, l'implémentation des matrices peut être modifiée simplement en réécrivant les primitives d'accès aux matrices (aux tableaux de résultats). Rappelons que les procédures ou fonctions à réécrire dans ce cas sont notées "♦♦♦" dans les spécifications (voir annexe 1).

6.2 Le simulateur

Tout simulateur se base sur un calculateur et sur un tableau de résultats pour calculer et mémoriser, à chaque observation, les valeurs que prennent les différentes variables. Le simulateur et le calculateur sont tous deux des programmes au sens informatique du terme.

Le langage 'turbo-Pascal' ne permet pas de fournir une procédure en tant que paramètre à une autre procédure. Or, lors d'une simulation, le simulateur doit appeler le calculateur choisi (écrit) par le constructeur du simulateur. Il n'est donc pas possible de prévoir une seule procédure (simulateur) permettant de simuler n'importe quelle expérience, c'est-à-dire se basant sur n'importe quel calculateur.

Nous avons donc écrit un simulateur typique, appelons-le simulateur standard, faisant appel à un calculateur fictif (appelé "calculateur"). La personne désirant construire un nouveau simulateur doit recopier le texte du simulateur standard en remplacant le nom du calculateur qui y figure par le nom réel du calculateur qu'elle a choisi (ou écrit). En outre, certaines modifications du texte du simulateur standard doivent être effectuées en fonction du nombre d'arguments du calculateur (voir manuel du constructeur du simulateur, p. 93).

Le nombre de paramètres du calculateur ne peut dépasser la constante 'largmax'. Le ième paramètre du calculateur est noté p[i]. Chacun de ces paramètres est en fait stocké dans un vecteur de réels de longueur 'largmax', appelé 'p'. L'élément p[i] fait donc référence au ième de ses éléments.

Il est à noter que la procédure 'initsimul' à laquelle fait appel le simulateur standard vérifie dans la mesure du possible la cohérence des arguments fournis au simulateur. Elle s'assure notamment que le nombre de variables définies dans le tableau de résultats est égal au nombre de paramètres du calculateur. Comme ce nombre dépend du calculateur, il devra être spécifié par le constructeur du simulateur qui recopie le texte du simulateur standard. Il s'agit du dernier argument de la procédure 'initsimul', qui est un entier.

Si le lecteur désire en savoir plus à propos des paramètres ou consulter le texte du simulateur, il peut se référer aux paragraphes relatifs à la construction du simulateur (p. 93) et à l'utilisation de celui-ci (p. 97).

Comme nous l'avons déjà expliqué ci-dessus, le calcul et l'utilisation des variables indépendantes non connues a priori n'a pas été implémenté, mais a été prévu au niveau du simulateur. Pour implémenter ce mode de détermination des variables indépendantes, il suffit d'écrire la procédure 'calculvinc' en respectant les spécifications qui en sont données (voir annexes), et en se basant sur les renseignements repris dans la définition de telles variables.

Afin d'éviter la création manuelle des simulateurs, il serait ultérieurement possible d'automatiser la génération des simulateurs désirés. Il suffirait de réaliser un préprocesseur qui, à partir du programme source, effectuerait toutes les éditions nécessaires.

6.3 Remarques

Certaines procédures de gestion d'entrées-sorties utilisent le concept du nombre de colonnes et de lignes de l'écran. Pour assurer la portabilité du système, nous avons isolé ces deux valeurs dans les constantes 'maxcolecran' et 'maxlignecran'. Ces constantes sont faciles d'accès, car placées au début du texte du progiciel, et donc aisément modifiables.

La procédure 'lirestring' reçoit comme paramètre une chaîne de caractères déclarée de type 'stringmax'. La chaîne de caractères 'stringmax' est déclarée de longueur égale à une constante 'longstring'. Cette constante est égale à 50 dans ce prototype. Il est par conséquent aisé de modifier la longueur de la chaîne de caractères de type 'stringmax'.

Nous donnons en annexe les spécifications des procédures et fonctions implémentées dans le progiciel (annexe 1). L'architecture physique de ces primitives est également disponible à l'annexe 2.

CHAPITRE 7 : MANUELS UTILISATEURS

7.1 Manuel du constructeur du simulateur

Il n'est pas possible, pour des raisons liées au langage utilisé, de prévoir dans le cadre de ce travail une seule et unique procédure qui jouerait le rôle de simulateur, et à laquelle le calculateur serait passé comme argument. Par conséquent, tout simulateur aura un texte de programme propre.

Dans le but d'aider le constructeur du simulateur, nous avons créé un simulateur que nous appelons simulateur standard. Ce programme fait appel à certaines procédures ou fonctions du progiciel. Ces procédures et fonctions sont: 'initsimul', 'lgmat', 'lirepi' et 'ecrirepi'. Le constructeur du simulateur qui désire générer un nouveau simulateur doit recopier le texte du simulateur standard en l'adaptant au calculateur choisi. Les adaptations à faire sont les suivantes :

- rebaptiser le simulateur
- remplacer le nom du calculateur standard par le nom du calculateur réellement appelé
- mettre à jour les paramètres du calculateur
- mettre à jour le dernier argument de la procédure initsimul appelée par le simulateur.

Le nombre de paramètres du calculateur ne peut dépasser 20. Cette limite est fixée arbitrairement dans ce prototype. Il est possible de la modifier (voir choix d'implémentation, p. 88). Dans le texte du simulateur, le nombre de paramètres à fournir au calculateur doit être mis à jour en fonction du calculateur choisi. Si le calculateur a n paramètres, ceux-ci doivent être écrits p[1], p[2], ...p[i]...p[n] (voir figure 7.1).

La dernière modification à apporter par le constructeur de simulateur au texte du simulateur standard est la mise à jour du dernier argument de la procédure 'initsimul'. La procédure 'initsimul' vérifie dans la mesure du possible la cohérence des arguments fournis au simulateur. Elle s'assure entre autres qu'il y a autant de paramètres fournis

au calculateur que de variables définies dans le tableau de résultats utilisé par le simulateur. Le nombre de paramètres du calculateur est connu du constructeur du simulateur. Ce nombre doit être affecté au dernier des arguments de la procédure 'initsimul'.

Les deux paramètres du simulateur sont un tableau de résultats de type 'tabres', et un vecteur de noms de variable de type 'vectstring'. Ces deux types sont définis au sein du progiciel. Ce second argument est nécessaire car l'ordre des variables définies dans le tableau de résultats ne doit pas forcément être le même que l'ordre des paramètres du calculateur, paramètres qui représentent aussi des variables. Le deuxième argument du simulateur reprend tous les noms de variables définies dans le tableau de résultats mais retriés dans l'ordre correspondant aux paramètres du calculateur. Ceci permet au simulateur de savoir comment mettre en relation les variables du tableau de résultats avec les paramètres du calculateur. Un exemple de telle correspondance est donné plus loin (voir utilisation du simulateur, p. 97).

Les deux arguments sont fournis au simulateur par son utilisateur, c'est-à-dire le constructeur du didacticiel. Nous reparlerons de ces paramètres dans le manuel destiné au constructeur du didacticiel.

Les spécifications du simulateur standard sont les suivantes:

arg : tab : tabres
chaine : vectstring

res : tab : tabres
messages d'erreur

specif : si le vecteur-définition ou la matrice de 'tab' est vide, ou
si à une variable indépendante connue a priori correspond une colonne vide dans la matrice, ou
si le vecteur 'chaine' ne comprend pas autant de noms qu'il y a de variables définies dans 'tab', ou
si le vecteur 'chaine' ne comprend pas les mêmes noms que les variables définies dans le vecteur-définition de 'tab', ou
si le nombre de paramètres du calculateur est différent du nombre de variables définies dans 'tab', ou
si il n'est pas possible d'ajouter à la matrice de 'tab' le nombre de colonnes nécessaires pour qu'il y ait autant de colonnes que de variables définies dans 'tab',

alors afficher un message d'erreur
sinon pour chaque ligne de la matrice de 'tab':
- calculer et inscrire dans la matrice (à l'endroit correspondant) les valeurs des variables évolutives,
- appeler le calculateur et lui fournir les valeurs des différentes variables,
- écrire dans la matrice de 'tab' les valeurs des variables dépendantes et évolutives fournies par le calculateur.

Notons que dans le prototype du progiciel, les variables indépendantes non connues a priori ne sont pas implémentées.

Nous donnons ci-dessous le texte du simulateur standard. Ce qui doit être adapté lors de la création de nouveaux simulateurs est souligné. Nous donnons ensuite le texte d'un nouveau simulateur, appelé 'simul1' qui utilise le calculateur 'calcul1' (3 paramètres), dans lequel les adaptations ont été faites (voir figure 7.1).

Il est bien évident que le texte de la procédure 'calcul1', contenant trois paramètres, doit apparaître dans le texte du didacticiel au dessus de la procédure 'simul1'.

Remarquons que, dans le texte du calculateur, les variables évolutives et dépendantes doivent être considérées comme des paramètres par variable et non par valeur. Ces paramètres seront donc précédés de "var".

Une bibliothèque de simulateurs peut être constituée. Tout simulateur destiné à être classé dans une bibliothèque doit être accompagné d'une documentation rédigée par son créateur, et destinée à l'utilisateur du simulateur, c'est-à-dire un constructeur de didacticiel. Dans cette documentation doit être expliquée l'expérience qui est simulée, et quelle variable représente chacun des paramètres du caculateur. Il doit aussi y être fait mention de l'unité, du genre (et éventuellement du mode de détermination et des règles de calcul s'il s'agit de variable indépendante) de chaque variable.

texte du simulateur standard:

```
procedure simulateur (var t:tabres; chaine:vectstring);

   var l:integer;
       ok:boolean;

begin
  initsimul (chaine, t, ok, 20);
  if ok
    then for l:=1 to lgmat(t) do
         begin
         lirepi (chaine, t, l);
         calculateur (p[1], p[2], p[3], ..., p[20]);
         ecrirepi (chaine, t, l)
         end
end;
```

exemple de simulateur:

```
procedure simul1 (var t:tabres; chaine:vectstring);

   var l:integer;
       ok:boolean;

begin
  initsimul (chaine, t, ok, 3);
  if ok
    then for l:=1 to lgmat(t) do
         begin
         lirepi (chaine, t, l);
         calcul1 (p[1], p[2], p[3]);
         ecrirepi (chaine, t, l)
         end
end;
```

figure 7.1

7.2 Manuel du constructeur du didacticiel

7.2.1 Utilisation du simulateur

La documentation accompagnant obligatoirement tout simulateur renseigne l'utilisateur sur le calculateur auquel fait appel le simulateur.

Les deux paramètres d'un simulateur sont un tableau de résultats et un vecteur de noms de variables. Les types Pascal correspondant à ces deux arguments sont respectivement 'tabres' et 'vectstring', types prédéfinis dans le progiciel. Les paramètres passés à tout simulateur devront donc avoir été déclarés en utilisant ces noms de type prédéfinis.

Il est à noter que l'ordre des variables définies dans le tableau de résultats ne doit pas forcément être le même que l'ordre des paramètres du calculateur. Prenons un calculateur à trois paramètres et supposons par exemple que le premier paramètre du calculateur représente le volume, le deuxième la pression et le troisième la température. Ces trois variables peuvent avoir été définies dans un ordre différent dans le tableau de résultats. Supposons que la première variable définie soit la pression sous le nom 'press', la deuxième la température sous le nom 'Temp' et la dernière le volume sous le nom 'vol'. Ceci explique le rôle du second argument du simulateur, de type 'vectstring'. C'est un vecteur reprenant tous les noms des variables définies dans le tableau de résultats mais retriées dans l'ordre correspondant aux paramètres du calculateur. Dans l'exemple ci-dessus, ce vecteur serait :'vol', 'press', 'Temp'.

Des outils de construction de vecteurs de noms de variables sont mis à la disposition du constructeur de didacticiel. Avant d'y inscrire un seul nom de variable, tout vecteur de nom de variable doit être initialisé (procédure 'initchaine'). L'inscription d'un nom de variables dans un tel vecteur se fait par l'utilisation de la procédure 'ajoutnom'.

Un certain nombre de conditions, portant sur les arguments du simulateur, doivent être satisfaites pour que la simulation ait lieu. Si au moins une de ces conditions n'est pas satisfaite, un message d'erreur apparaît à l'exécution du didacticiel et précise quelle condition n'est pas satisfaite.

Les conditions à satisfaire pour que la simulation ait lieu sont :

- le vecteur-définition et la matrice du tableau de résultats ne doivent pas être vides;
- toute colonne de la matrice correspondant à une variable définie comme indépendante connue a priori dans le tableau de résultats ne doit pas être vide;
- le vecteur de nom de variable doit comprendre autant de noms qu'il y a de variables définies dans le tableau de résultats;
- le vecteur de nom de variable doit comprendre les mêmes noms que les variables définies dans le tableau de résultats;
- il doit y avoir autant de variables définies dans le tableau de résultats que de paramètres au calculateur.

Pour plus d'informations sur le simulateur, le lecteur peut consulter les spécifications du simulateur standard (p. 94).

7.2.2 Utilisation des outils du progiciel

Ont été implémentés dans le prototype du progiciel et mis à la disposition du constructeur de didacticiel des outils de création, manipulation et visualisation de tableaux de résultats, ainsi que quelques primitives d'entrée-sortie.

Tout constructeur de didacticiel peut faire appel à des procédures ou fonctions du progiciel en inscrivant la phrase ($I 'nom fichier') au début du texte de son didacticiel, en remplacant 'nom fichier' par le nom externe du fichier dans lequel se trouve le progiciel.

Au début de son programme, le constructeur du didacticiel doit effectuer un certain nombre d'initialisations; ceci se fait par appel de la procédure 'initialisation'. De plus, tout tableau de résultats déclaré (de type 'tabres') doit être initialisé. La procédure 'inittab' se charge d'effectuer cette initialisation. Ces diverses initialisations sont obligatoires.

Les arguments de la plupart des procédures et fonctions du progiciel doivent répondre à certaines conditions. Ces procédures (ou fonctions) vérifient dans la mesure du possible, la cohérence de leurs arguments. Si une condition portant sur un de ses arguments n'est pas satisfaite,

l'exécution de la procédure aura pour unique effet l'affichage d'un message d'erreur précisant la cause de cette erreur. Ces tests effectués par la plupart des procédures (fonctions) à propos de la cohérence des arguments sont destinés à aider le programmeur à mettre au point son didacticiel. Après la lecture du message d'erreur affiché lors de l'exécution de son programme, le constructeur de didacticiel doit presser n'importe quelle touche pour que l'exécution continue.

C'est également dans le but d'aider le constructeur de didacticiel que tout message d'erreur contiendra le nom de la procédure (ou fonction) qui envoie le message d'erreur. L'utilisateur peut localiser ainsi facilement son erreur.

Dans le cadre de ce prototype, le nombre de variables qui peuvent être définies dans un vecteur-définition, ou le nombre de noms que peut contenir un vecteur de noms de variables (vectstring) ne doit pas etre supérieur à 20.

En ce qui concerne les matrices, le nombre de colonnes ne peut non plus être supérieur à 20, tandis que le nombre de lignes n'a pas de borne fixe. A priori, aucune restriction ne porte sur le nombre de lignes d'une matrice. Il se peut toutefois que la place maximale réservée pour les matrices soit entièrement occupée. Ce peut être le cas lorsque beaucoup de tableaux sont manipulés en même temps, surtout si les matrices qui y sont associées contiennent beaucoup de lignes. L'utilisateur peut libérer de la place en remettant à vide des matrices qui ne sont plus exploitées par la suite, dans le didacticiel. La procédure 'liberemat' est prévue à cet effet.

Si le constructeur du didacticiel veut s'assurer qu'il est possible d'attribuer à un tableau de résultats particulier une matrice d'un nombre de lignes et de colonnes donné, il peut utiliser la fonction 'assezplace'.

Les procédures 'defevol', 'defdep' et 'deficon' permettent de créer des définitions de variables respectivement évolutives, dépendantes et indépendantes connues a priori. Les variables indépendantes non connues à priori ne sont pas implémentées au niveau de ce prototype. Le nom d'une variable ne peut dépasser 8 caractères, et l'unité 10 caractères. Ces valeurs sont fixées arbitrairement, et peuvent facilement être modifiées (voir implémentation des vecteurs-définition p. 88).

Les procédures 'ajoutdef' et 'changedef' permettent de construire ou modifier des vecteurs-définition. Une matrice à une colonne peut être créée et remplie par la primitive 'creerpas'.

Il est très important de noter que les matrices et les vecteurs-définition ne sont identifiés que par les noms des tableaux de résultats dont ils font partie (cfr concepts relatifs aux manipulations de tableaux de résultats).

Il est possible de constituer un nouveau vecteur-définition par manipulation d'autres vecteurs-définition, sans que la matrice correspondant à ce nouveau vecteur n'en soit affecté. L'inverse est également vrai: il est possible de construire une nouvelle matrice sans affecter le vecteur-définition correspondant. D'autres procédures sont mises à la disposition de l'utilisateur pour construire à la fois un nouveau vecteur-définition et une nouvelle matrice faisant tous deux partie du même tableau de résultats (cfr concepts relatifs aux manipulations de tableaux).

Les procédures dont les noms se terminent par "-vect" travaillent uniquement sur des vecteurs-définition. Celles qui se terminent par "-mat" travaillent sur des matrices uniquement. Les procédures dont le suffixe est "-tab" travaillent à la fois sur la matrice et sur le vecteur-définition d'un même tableau de résultats.

Le type de chaque argument des primitives est précisé dans les spécifications (annexe 1). La plupart de ces types sont prédéfinis dans le système. L'utilisateur n'a donc plus à les définir; il doit les utiliser tels quels. La liste de ces types prédéfinis est donnée ci-dessous. La colonne de droite précise à quoi correspondent ces types.

tabres	tableau de résultats
setchar	ensemble (set) de caractères
variable	définition de variable
stringmax	chaîne de caractères (longueur maximale = 50)
strnm	chaîne de caractères (longueur maximale = 8)
strut	chaîne de caractères (longueur maximale = 10)
vectstring	vecteur de noms de variable (strnm) (longueur maximale = 20)

Une liste reprenant les mots déjà utilisés dans le progiciel est donnée en annexe (voir liste de mots réservés). Ces mots ne peuvent plus être utilisés dans un programme faisant appel au progiciel, pour déclarer des constantes, types ou variables.

La construction d'un didacticiel est schématisé à la figure 7.2.

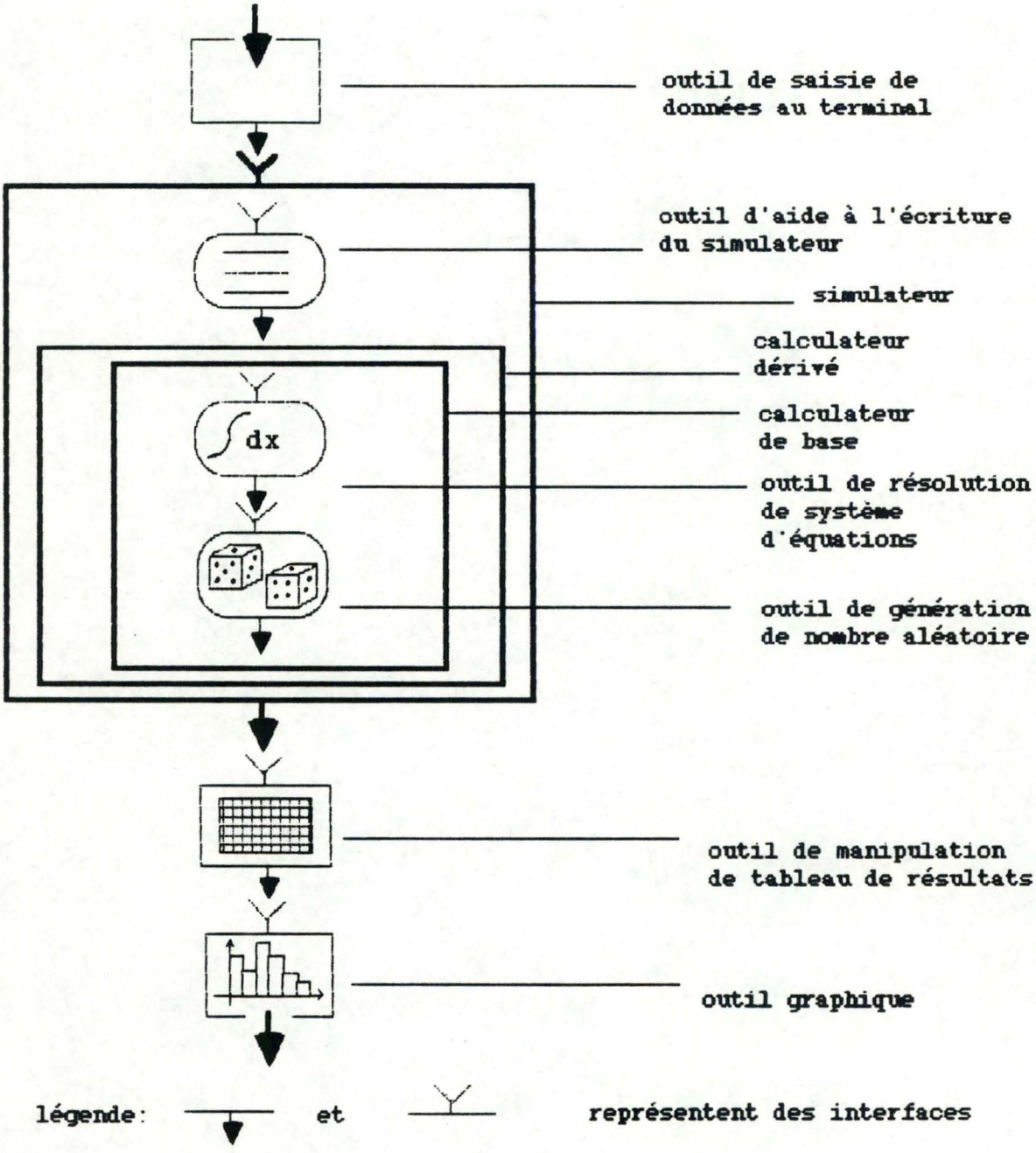


figure 7.2

Nous reprenons ci-dessous les différentes procédures et fonctions à la disposition du constructeur du didacticiel en les accompagnant d'une brève description. Si l'utilisateur désire plus de précisions, il peut se référer aux spécifications exactes de la primitive (le numéro de page auquel il doit se reporter est indiqué).

Procédure **INITIALISATION** (p. 113)

initialise le système

Procédure **INITTAB** (p. 113)

initialise un tableau de résultats

Procédure **INITCHAINE** (p. 116)

initialise un vecteur de noms de variable

Procédure **AJOUTNOM** (p. 116)

ajoute un nom de variable à un vecteur de noms de variable donné

Procédure **DEFEVOL** (p. 117)

crée une définition de variablle évolutive

Procédure **DEFDEP** (p. 117)

crée une définition de variable dépendante

Procédure **DEFICON** (p. 117)

crée une définition de variable indépendante connue à priori

Procédure **AJOUTDEF** (p. 115)

ajoute une définition de variable à un vecteur-définition donné

Procédure **CHANGEDEF** (p. 115)

modifie, au sein d'un vecteur-définition donné, la définition d'une variable

Procédure **CREERPAS** (p. 125)

remplit une matrice à une colonne par les valeurs générées à partir d'une borne inférieure jusqu'à une borne supérieure, par progression donnée.

Procédure **LIBEREMAT** (p. 119)

met à vide une matrice donnée

Fonction **ASSEZPLACE** (p. 120)

permet de savoir s'il est possible de créer une matrice de dimensions données.

Procédure **COPIERMAT** (p. 129)

recopie une matrice dans une autre matrice

Procédure **COPIERVECT** (p. 130)

recopie un vecteur-définition dans un autre vecteur-définition

Procédure **COPIERTAB** (p. 131)

recopie un tableau de résultats dans un autre tableau de résultats

Procédure **JUXTAMAT** (p. 126)

crée une matrice par juxtaposition de deux autres matrices

procédure **JUXTAVECT** (p. 127)

crée un vecteur-définition par juxtaposition de deux autres vecteurs-définition

Procédure **JUXTATAB** (p. 128)

crée un tableau de résultats par juxtaposition de deux autres tableaux de résultats

Procédure **PRODMAT** (p. 132)

crée une matrice par produit cartésien ordonné de deux autres matrices

Procédure **PRODTAB** (p. 133)

crée un tableau de résultats par produit cartésien ordonné de deux autres tableaux de résultats

Procédure **AFFICHEMAT** (p. 121)

affiche les valeurs reprises dans une matrice donnée

Procedure **LIRESTRING** (p. 118)

lit une chaîne de caractères au terminal et n'accepte que les caractères faisant partie d'un ensemble de caractères dits acceptables

Procedure **LIRENOMBRE** (p. 119)

lit un réel au terminal

7.3 Exemple d'utilisation du système

Nous présentons en annexe le texte d'un didacticiel mis au point à l'aide des outils de notre système.

Ce programme utilise le simulateur 'simulat1'. Le simulateur quant à lui fait appel au calculateur 'calcul1' dérivé du calculateur 'biochem'. Les textes de ces différents programmes sont également disponible en annexe (annexe 5).

Le didacticiel que nous avons développé s'inspire d'un modèle biochimique qui permet de simuler l'évolution dans le temps de l'équilibre entre quatre composants d'une réaction. L'équation de la réaction est de la forme :

$$A + B \underset{k2}{\overset{k1}{\longleftrightarrow}} C + D$$

où A, B, C, et D représentent les concentrations des différents réactifs, et k1 et k2 sont les constantes de taux de réaction.

Le didacticiel lit au terminal les concentrations originales des différents composants, ainsi que les valeurs des constantes k1 et k2. L'intervalle de temps entre deux observations et le temps total de l'expérience à simuler sont aussi précisés par l'élève.

Le nombre maximal d'observations qu'il est possible d'effectuer est de 500. Le temps total à introduire doit donc être choisi en tenant compte de l'intervalle de temps sélectionné et du nombre maximal d'observations possibles.

Après la simulation, les résultats des différentes observations sont visualisés sous forme d'un tableau à deux dimensions. Pour chaque observation simulée, les valeurs des différentes concentrations, ainsi que le temps, sont affichés sur une même ligne du tableau.

CONCLUSION

L'objectif de ce projet est de proposer un système constitué d'un ensemble intégré d'outils destinés à aider les enseignants du secondaire à mettre au point un didacticiel. Si ce système est destiné à l'enseignement de la biologie, il est aussi utilisable dans tout contexte où des modèles numériques sont employés (chimie, physique, voir même psychologie et économie).

Dans le soucis de laisser notre système le plus ouvert possible, un certain nombre de concepts ont été isolés et définis. Nous avons surtout analysés les concepts relatifs:

- au simulateur qui gère la simulation d'une série d'observations sur base d'un modèle mathématique
- aux tableaux de résultats qui mémorisent les résultats des simulations
- à la visualisation graphique de ces résultats.

La description de ces concepts nous a permis de proposer un certain nombre d'outils cohérents.

Un prototype de ce système a été implémenté. Celui-ci gère la mémorisation des tableaux de résultats, et propose des outils de construction de simulateur et de manipulation des tableaux de résultats. Il offre également quelques primitives d'entrée-sortie.

Afin de juger de l'utilité de notre prototype, nous avons mis au point un didacticiel de démonstration.

Certains concepts dégagés de l'analyse nous ont permis de mettre en évidence d'autres outils que ceux repris dans notre prototype. Ceux-ci doivent donc encore être implémentés. C'est entre autres le cas des outils graphiques et de saisie de données.

Au sein d'un didacticiel, le dialogue avec l'élève tient une place non négligeable. C'est pourquoi les entrées-sorties en général pourraient faire l'objet d'une analyse plus détaillée. Ceci permettrait de proposer des outils plus perfectionnés encore.

Il serait aussi intéressant de constituer une bibliothèque de modèles. Cette bibliothèque serait à la disposition des constructeurs de simulateur. Dans le même ordre d'idée, on pourrait envisager la constitution d'une

bibliothèque de méta-modèles (résolution de système d'équations linéaires, différentielles,...).

D'autre part, il est possible de rendre le système plus transparent à l'utilisateur en concevant un langage d'auteur. Celui-ci ferait appel au système par le truchement d'un précompilateur.

Il est important d'expérimenter un système comme le nôtre. Il faudrait, par conséquent, le placer entre les mains d'enseignants du secondaire, en vue de juger de son utilité et de son efficacité.

Au terme de ce travail, nous n'avons pas la prétention d'affirmer que le système que nous avons défini se suffit à lui-même. La porte est donc laissée ouverte à toute proposition d'outils qui vont dans le sens de nos objectifs.

REFLEXIONS PERSONNELLES

J'aimerais résumer en quelques lignes mes impressions à propos de ce mémoire.

Ce travail m'a beaucoup intéressée car il porte sur deux disciplines qui sont pour moi des domaines de prédilection, à savoir la biologie et l'informatique.

Ma formation en biologie m'avait habituée à la démarche de type expérimentale ou inductive. C'est pourquoi j'ai éprouvé certaines difficultés au début de ce travail, car la démarche y était plutôt de type déductive.

De plus, de par ma première formation, je n'avais pas l'esprit d'analyse nécessaire en informatique. Ce mémoire m'aura en bonne partie aidée à l'acquérir.

D'autre part, j'aimerais que mon enthousiasme pour le sujet traité dans ce mémoire soit partagé. Je souhaite que quelqu'un ait à coeur de reprendre le travail entamé ici.

ANNEXE 1 : SPECIFICATIONS

Tout message d'erreur à afficher est spécifique de l'erreur commise et, par conséquent, doit permettre au lecteur du message d'identifier l'erreur. La plupart des messages d'erreur reprennent le nom de la procédure (ou fonction) qui a été utilisée et dont les conditions portant sur les arguments n'ont pas étés satisfaites. Le nom de la procédure (ou fonction) y apparaît entre parenthèses.

Les messages d'erreur destinés à l'utilisateur final (élève) ne reprennent pas le nom de la procédure (ou fonction) dont ils proviennent. Il s'agit surtout de primitives gérant des entrées-sorties. C'est notamment le cas de la procédure "lirenombre" qui lit une suite de caractères introduits au clavier. Si cette suite de caractères ne représente pas un réel, un message avertit l'élève et lui demande de recommencer.

Les différents types associés aux arguments et résultats dans les spécifications ci-dessous correspondent à des types au sens Pascal. La correspondance entre les noms de types employés dans les spécifications et ce qu'ils représentent est reprise ci-dessous :

tabres	tableau de résultats
setchar	ensemble (set) de caractères
variable	définition de variable
stringmax	chaine de caractères (longueur maximale = 50)
strnm	chaine de caractères (longueur maximale = 8)
strut	chaine de caractères (longueur maximale = 10)
vectstring	vecteur de noms de variable (strnm)
vectcol	vecteur d'entiers
vectval	vecteur de réels

ces trois types de vecteurs ont une longueur maximale egale à 'longmax' qui est une variable globale

Les matrices et vecteur-définition sont identifiés par les noms des tableaux de résultats dont ils proviennent. Le nombre maximal de variables que peut contenir un tableau de résultats est égal à une constante globale 'largmax'.

Remarquons encore qu'une distinction est faite entre la longueur logique et physique des vecteurs-définitions, et le nombre logique et physique de lignes d'une matrice. La justification de cette distinction est donnée dans le chapitre 6.

Les procédures et fonctions notées ♦♦♦ doivent être réécrites si les tableaux de résultats sont implémentés différemment.

Il est possible à un constructeur d'outils de mettre au point ultérieurement de nouveaux outils en faisant appel à certaines procédures ou fonctions existant dans le système. Si ces outils manipulent des tableaux de résultats, et que l'implémentation de ces derniers n'est pas modifiée, une contrainte devra alors être respectée. Les tableaux de résultats qui sont des paramètres pour ces nouveaux outils doivent être des paramètres par variable et non par valeur. Ceci est obligatoire si le nouvel outil fait appel à une procédure ou fonction du système recevant, elle-aussi, un tableau de résultats comme paramètre. Cette contrainte est due à l'implémentation des tableaux de résultats dans ce prototype.

Illustrons ceci par un exemple. Supposons que le nouvel outil, appelons-le 'supertab', fait appel à la procédure 'copiertab' implémentée dans notre système : l'écriture de la nouvelle procédure est ébauchée ci-dessous :

```
procedure supertab (var tab1,tab2 : tabres ...)

begin
  .
  .
  copiertab (tab1, tab2)
  .
end
```

Procédure **LIRE** (tab, col, lig, res) ♦♦♦

argum. : tab : tabres
col, lig : entier

résul. : res : réel

précond. : . 'tab' contient une matrice non vide
. 'col' est compris entre 1 et le nombre (logique) de colonnes de la matrice de 'tab'
. 'lig' est compris entre 1 et le nombre (logique) de lignes de la matrice de 'tab'

spécif. : renvoyer la valeur 'res' se trouvant à la colonne n·'col' et la ligne n·'lig' de la matrice de 'tab'

est appelée par : lirepi, juxtaval, copierval, prodval, affichemat

Procédure **ECRIRE** (tab, col, lig, val) ♦♦♦

argum. : tab : tabres
val : réel
col, lig : entiers

résul. : tab : tabres

précond. : . 'tab' contient une matrice non vide
. 'col' est compris entre 1 et le nombre logique de colonnes de la matrice de 'tab'
. 'lig' est compris entre 1 et le nombre logique de lignes de la matrice de 'tab'

spécif. : écrire la valeur 'val' à la colonne n·'col' et à la ligne n·'lig' de la matrice de 'tab'

est appelée par : lirepi, écrirepi, créerpas, juxtaval, copierval, prodval

Fonction **LGMAT** (tab): entier ♦♦♦

argum. : tab : tabres

spécif. : est égal au nombre (logique) de lignes de la matrice de 'tab'

est appelée par : simulateur, juxtaval, juxtamat, juxtatab, copierval, prodval, affichemat, copiermat, prodmat, prodtab, copiertab

Fonction **LRMAT** (tab): entier ♦♦♦

argum. : tab : tabres

spécif. : est égal au nombre (logique) de colonnes de la matrice de 'tab'

est appelée par : coherence, elargir, initsimul, juxtaval, juxtamat, juxtatab, copierval, prodval, affichemat, copiermat, copiertab, prodval, prodmat, prodtab

Fonction **VECTLG** (tab): entier ♦♦♦

argum. : tab : tabres

spécif. : est égal au nombre (logique) de définitions de variables contenues dans le vecteur-définition de 'tab'

est appelé par : memenom, existnom, ajoudef, coherence, initsimul, juxtadef, juxtavect, juxtatab, copierdef, copiervect, copiertab, prodtab

Procédure **IEMEDEF** (tab, def, i) ♦♦♦

argum. : tab : tabres
i : entier

résul. : def : variable

précond. : 'i' est compris entre 1 et le nombre (logique) de définitions contenues dans le vecteur-définition de 'tab'

spécif. : renvoyer la définition de variable 'def' se trouvant à la position 'i' du vecteur-définition de 'tab'

est appelée par : memenom, existnom, initsimul, lirepi, ecrirepi, juxtadef, copierdef, prodtab, affichemat

Fonction **VECTVIDE** (tab): booléen ♦♦♦

argum. : tab : tabres

spécif. : est vrai si le vecteur-définition de 'tab' ne contient aucune définition de variable, est faux sinon

est appelée par : changedef, coherence, initsimul, juxtadef, juxtavect, copierdef, copiervect, copiertab

Procédure **INITIALISATION** ♦♦♦

spécif. : initialise le système

Procédure **INITVECT** (tab) ♦♦♦

résul. : tab : tabres

spécif. : mettre à vide le vecteur-définition de 'tab'

est appelée par : inittab, juxtadef, copierdef

Procédure **INITTAB** (tab) ♦♦♦

résul. : tab : tabres

spécif. : mettre à vide le vecteur-définition et la matrice de 'tab'

appelle : initvect

Fonction **VERIFLGVECT** (tab, i) : booléen ♦♦♦

argum. : tab : tabres
i : entier

spécif. : est vrai si le vecteur-définition de 'tab' peut contenir au moins 'i' définitions de variable, est faux sinon

est appelée par : ajoudef, juxtavect, juxtatab, copiervect, copiertab, prodtab

Procédure **ECRIREDEF** (tab, def, i) ♦♦♦

argum. : def : variable
i : entier

résul. : tab : tabres

précond. : 'i' est compris entre 1 et la longueur (logique) du vecteur-définition de 'tab'

spécif. : écrire la définition 'def' à la ième position dans le vecteur-définition de 'tab'

est appelée par : ajoudef, changedef, copierdef, juxtadef

Fonction **POSITION** (tab, nom): entier ♦♦♦

argum. : tab : tabres
nom : strnm

précond. : 'nom' est différent de la chaîne vide

spécif. : est égal à la position, dans le vecteur-définition de 'tab', de la définition de variable dont le nom est 'nom' si cette définition s'y trouve, est égal à zéro sinon

est appelée par : changedef

Fonction **MEMENOM** (tab1, tab2): booléen

argum. : tab1, tab2 : tabres

spécif. : est vrai si les vecteurs-définition de 'tab1' et de 'tab2' contiennent la définition d'une variable portant le même nom, est faux sinon

est appelée par : juxtavect, juxtatab, prodtab

appelle : vectlg, iemedef

Fonction **EXISTNOM** (tab, nom): booléen

argum. : tab : tabres
nom : strnm

spécif. : est vrai si le vecteur-définition de 'tab' contient la définition d'une variable s'appelant 'nom', est faux sinon

est appelée par : ajoudef, changedef, initsimul

appelle : vectlg, iemedef

Procédure **AJOUTDEF** (tab, def)

argum. : def : variable

précond. : le nom repris dans la définition de variable 'def' est différent de la chaîne vide

résul. : tab : tabres
messages d'erreur

spécif. : s'il est impossible d'ajouter une définition de variable au vecteur-définition de 'tab',
ou si le nom repris dans la définition 'def' est un nom de variable déjà existant dans le vecteur-définition de 'tab'
alors afficher un message d'erreur,
sinon inscrire la définition 'def' dans le vecteur-définition de 'tab' après la dernière définition de variable qui y figure déjà (si le vecteur-définition de 'tab' ne comprend encore aucune définition, inscrire 'def' en première position)

appelle : veriflgvect, existnom, ecriredef, vectlg

Procédure **CHANGEDEF** (tab, nom, def)

argum. : nom : strnm
def : variable

résul. : tab : tabres
messages d'erreur

spécif. : si le vecteur-définition de 'tab' est vide
ou si 'nom' est une chaîne vide
ou si 'nom' n'est pas le nom d'une variable

définie dans le vecteur-définition de 'tab',
ou si le nom repris dans 'def' est déjà un nom de variable définie dans le vecteur-définition de 'tab' (le nom de 'def' peut cependant être égal à 'nom')
alors afficher un message d'erreur,
sinon remplacer dans le vecteur-définition de 'tab' la définition de la variable de nom 'nom' par la définition 'def'

appelle : ecriredef, vectvide, existnom, position

Fonction **EXIST** (tab): booléen ♦♦♦

argum. : tab : tabres

spécif. : est vrai si la matrice de 'tab' est vide, est faux sinon

est appelée par : liberemat, creerpas, juxtamat, juxtatab, copiermat, copiertab, prodtab, prodmat, affichemat, initsimul

Procédure **AJOUTNOM** (chaine, nom)

argum. : nom : strnm

résul. : chaîne : vectstring
messages d'erreur

spécif. : si 'nom' fait déjà partie de 'chaîne',
ou si 'nom' est une chaîne vide,
ou s'il est impossible d'ajouter un nouvel élément au vecteur 'chaîne' (plus de place)
alors afficher un message d'erreur,
sinon inscrire l'élément 'nom' après le dernier nom de variable (différent de la chaine vide) dans le vecteur 'chaîne'

Procédure **INITCHAINE** (chaine)

résul. : chaîne : vectstring

spécif. : mettre toutes les chaînes de caractères du vecteur 'chaîne' à vide

Procédure **DEFEVOL** (def, nom, unite, orig)

argum. : nom : strnm
unité : strut
orig : réel

résul. : def : variable
messages d'erreur

spécif. : si 'nom' est une chaîne vide, afficher un message d'erreur,
sinon créer 'def' qui est une définition de variable dont le nom est 'nom', l'unité est 'unité', le genre est "évolutive" et la valeur origine est 'orig'

Procédure **DEFDEP** (def, nom, unite)

argum. : nom : strnm
unité : strut

résul. : def : variable
messages d'erreur

spécif. : si 'nom' est une chaîne vide, afficher un message d'erreur,
sinon créer 'def' qui est une définition de variable dont le nom est 'nom', l'unité est 'unité' et le genre est "dépendante"

Procédure **DEFICON** (def, nom, unite)

argum. : nom : strnm
unité : strut

résul. : def : variable
message d'erreur

spécif. : si 'nom' est une chaîne vide, afficher un message d'erreur,
sinon créer 'def' qui est une définition de variable dont le nom est 'nom', l'unité est 'unité', le genre est "indépendante" et le mode de détermination est "connue (à priori)"

Fonction **COHERENCE** (tab): booléen

argum. : tab : tabres

spécif. : est vrai si le vecteur-définition et la matrice de 'tab' ne sont pas vides, et si le nombre (logique) de colonnes de la matrice est égal au nombre (logique) de définitions du vecteur-définition de 'tab'

appelle : vectvide, exist, vectlg, lrmat

est appelé par : juxtatab, prodtab

Procédure **LIRESTRING** (strg, x, y, long, okset)

argum. : x, y, long : entiers
okset : setchar

résul. : strg : stringmax

spécif. : si 'x' est inférieur à 1 ou supérieur au nombre de colonnes de l'écran,
ou si 'y' est inférieur à 1 ou supérieur au nombre de lignes de l'écran,
afficher un message d'erreur,
sinon . afficher 'long' fois le caractère "." à partir de la coordonnée-écran ('x','y': n° de colonne, n° de ligne);
. tant que le caractère de fin de ligne n'est pas introduit (<return>) et que le nombre de caractères introduits et acceptés ne dépasse pas 'long' :
- lire le caractère introduit
- n'afficher le caractère que s'il fait partie de l'ensemble des caractères acceptables 'okset'
- s'il s'agit de la touche "←—", effacer le dernier caractère introduit et affiché,
. constituer la chaîne de caractères 'strg' formée par la suite des caractères introduits et acceptés (et affichés)

rem : L'affichage des caractères acceptés se fait à partir de la coordonnée ('x','y') à l'écran.

est appelée par : lirenombre

Procédure **LIRENOMBRE** (nombre, x, y)

argum. : x, y : entiers

résul. : nombre : réel
messages d'erreur

spécif. : si 'x' n'est pas >= 1 et <= au nombre de colonnes de l'écran , ou si 'y' n'est pas >= 1 et <= au nombre de lignes de l'écran,
alors afficher un message d'erreur
sinon . afficher 16 fois le caractère "." à partir de la coordonnée-écran ('x','y'): (n· de colonne,n· ligne);
. tant que la suite de caractères (*) intoduite ne représente pas un réel acceptable par la machine, afficher un message d'erreur (au bas de l'écran: dernière ligne), sinon transformer la suite de caractère introduite en réel ('nombre')

(*) Les caractères introduits et ne faisant pas partie de la liste suivante : '0..9', '.', '+', '-', 'E','←' et le caractère de fin de ligne (return) ne sont ni affichés, ni considérés.
La touche '←' provoque l'effacement du dernier caractère introduit et affiché.
L'affichage des caractères introduits et acceptés se fait à partir de la coordonnée ('x','y') sur l'écran.

appelle : lirestring

Procédure **LIBEREMAT** (tab) ♦♦♦

résul. : tab : tabres
message d'erreur

spécif. : si la matrice de 'tab' n'est pas vide, afficher un message d'erreur,
sinon "mettre à vide" la matrice de 'tab'

appelle : exist

Procédure **CREERMAT** (tab, nbrelignes, nbecol, ok) ♦♦♦

argum. : nbrelignes, nbrecol : entiers

résul. : tab : tabres
ok : booléen
messages d'erreur

précond : la matrice de 'tab' est vide

spécif. : si 'nbrelignes' ou 'nbrecol' sont inférieurs ou égaux à zéro, ou s'ils sont (de manière générale) incohérents,
afficher un message d'erreur,
sinon . s'il est impossible d'attribuer à 'tab' une matrice de 'nbrelignes' lignes et 'nbrecol' colonnes (pas assez de place), alors afficher un message d'erreur et mettre ok à faux,
sinon attribuer à 'tab' une matrice de 'nbrelignes' lignes et 'nbrecol' colonnes et mettre ok à vrai.

est appelée par : creerpas, juxtamat, juxtatab, copiertab, copiermat, prodmat, prodtab

Fonction **ASSEZPLACE** (tab, lig, col): booléen ♦♦♦

argum. : tab : tabres
lig, col : entiers

condition : la matrice de 'tab' est vide

spécif. : est vrai s'il est possible d'attribuer à 'tab' une matrice contenant 'lig' lignes et 'col' colonnes, est faux sinon

Procédure **ELARGIR** (tab, i, bon) ♦♦♦

argum. : tab : tabres
i :entier

précond. : 'i' >= 0

résul. : tab : tabres
bon : booléen

spécif. : mettre 'bon' à faux s'il est impossible d'ajouter 'i' colonnes à la matrice de 'tab' (c'est-à-dire s'il est impossible d'attribuer à 'tab' une matrice plus large de 'i' colonnes que la matrice préexistante),
sinon mettre 'bon' à vrai et ajouter 'i' colonnes à la matrice de 'tab' (toutes les colonnes ont le même nombre de lignes)

appelle : lrmat

est appelée par : initsimul

Procédure **AFFICHEMAT** (tab)

argum. : tab : tabres
message d'erreur

spécif. : si la matrice de 'tab' n'existe pas, alors afficher un message d'erreur,
sinon . afficher dans l'ordre les nombres repris dans les 20 premières lignes et les 4 premières colonnes de la matrice de 'tab', si elle contient au moins autant de lignes et de colonnes, sinon afficher l'entièreté de la matrice;
. inscrire devant chaque ligne affichée le n· (logique) de cette ligne dans la matrice, et au-dessus de chaque colonne affichée le nom et l'unité de la variable correspondante (si cette variable a déjà été définie)
. tant que la touche <return> n'est pas frappée, permettre de visualiser les autres lignes et colonnes de la matrice si elles existent :
-si la touche '↑' est frappée, afficher dans l'ordre et si elles existent les 20 lignes précédant (dans la matrice) la première des lignes affichéesen dernier lieu (les colonnes sont les mêmes et les n· de lignes à imprimer à gauche de l'écran sont mis-à-jour)
-si la touche '←' est frappée, afficher dans l'ordre et si elles existent les 4 colonnes se trouvant (dans la matrice) à la gauche de la dernière des colonnes affichées en dernier lieu (les lignes sont les mêmes et les noms (et unités) de variables correspondant aux colonnes sont mis-à-jour)
-si la touche '→' est frappée,

afficher dans l'ordre le plus de colonnes possibles (max. 4) se trouvant (dans la matrice) à la droite de la première des colonnes affichées en dernier lieu (les lignes sont les mêmes et les noms (et unités) de variables correspondant aux colonnes sont mis-à-jour)

-si la touche ↓ est frappée, afficher dans l'ordre le plus de lignes possibles (max. 20) se trouvant (dans la matrice) en-dessous de la dernière des lignes affichées en dernier lieu (les colonnes sont les mêmes et les n· de lignes à imprimer à gauche de l'écran sont mis-à-jour)

rmq. : les nombres se trouvant sur une même ligne dans la matrice sont affichés sur une même ligne et dans le même ordre. Les nombres appartenant à une même colonne dans la matrice sont affichés sur une même colonne et dans le même ordre.

appelle : iemedef, lire, exist, lrmat, lgmat

Procédure **INITSIMUL** (chaine, tab, ok, x)

argum. : tab : tabres
chaîne : vectstring
x : entier

résul. : tab : tabres
ok : booléen
vectncol : vectcol (variable globale)
messages d'erreur

spécif. : si le vecteur-définition de 'tab' est vide,
ou si une variable définie comme indépendante et connue à priori dans le vecteur-définition de 'tab' correspond à une colonne vide de la matrice,
ou si le vecteur 'chaîne' ne comprend pas autant d'éléments que le vecteur-définition de 'tab',
ou si le vecteur 'chaîne' ne comprend pas les mêmes noms que les variables définies dans le vecteur-définition de 'tab' (ordre non significatif),
ou si 'x' n'est pas compris entre 1 et le nombre maximal d'éléments du vecteur chaine (= constante globale 'largmax'),
ou si le vecteur 'chaîne' et le vecteur-définition de 'tab' n'ont pas exactement 'x' éléments,

ou s'il n'est pas possible d'ajouter à la matrice de 'tab' le nombre de colonnes nécessaires pour qu'il y ait autant de colonnes que de définitions de variable dans 'tab' (toutes les colonnes ont le même nombre de lignes),
alors afficher un message d'erreur et mettre 'ok' à faux
sinon, pour chaque ième élément (soit A) du vecteur 'chaîne' : affecter au ième élément du vecteur 'vectncol' la position dans le vecteur-définition de 'tab' de la définition de variable dont le nom est A.

appelle : iemedef, lrmat, vectlg, existnom, elargir, vectvide, exist

est appelée par : simulateurs

Procédure **CALCULVINC** (tab, i, j)

argum. : tab : tabres
i, j : entiers

précond. : . la matrice de 'tab' n'est pas vide
. 'i' est compris entre 1 et le nombre (logique) de colonnes de la matrice de 'tab'
. 'j' est compris entre 1 et le nombre (logique) de lignes de la matrice de 'tab'

résul. : tab : tabres

spécif. : calculer la valeur de la variable indépendante non connue à priori à affecter à l'élément se trouvant à la 'i'ème colonne et 'j'ème ligne de la matrice de 'tab', et inscrire le résultat à cet endroit

est appelée par : lirepi

rmq. : non implémentée

Procédure **LIREPI** (chaine, tab, l)

argum. : chaîne : vectstring
vectncol : vectcol (variable globale)
tab : tabres
l :entier

précond. : . la matrice de 'tab' est non vide
. 'l' est compris entre 1 et le nombre (logique) de lignes de la matrice 'tab'
. 'chaîne' n'est pas vide
. 'chaîne' comprend autant d'éléments que le vecteur-définition de 'tab', et les noms repris dans 'chaîne' sont identiques aux noms des variables définies dans 'tab' (ordre non significatif)
. les valeurs de toutes les variables ont étés calculées et inscrites dans la matrice de 'tab' pour les lignes 1 à 'l'-1
. la valeur reprise dans le ième élément du vecteur 'vectncol' est égale à la position (c'est-à-dire au numero d'ordre) dans le vecteur-définition de 'tab' de la définition dont le nom est inscrit en ième position du vecteur 'chaine'

résul. : tab : tabres
p : vectval (variable globale)

spécif. : - pour chaque élément (soit A) du vecteur 'chaîne' : si A est le nom d'une variable définie dans 'tab' comme étant évolutive ou indépendante non connue (à priori)
alors calculer la valeur que prend cette variable à la ligne 'l' dans la matrice (avant l'exécution du calculateur) et l'inscrire dans la matrice à la ligne 'l' et à la colonne correspondante;
- pour chaque ième élément (soit A) du vecteur 'chaîne': écrire à la position i du vecteur 'p' la valeur que prend la variable dont le nom est A à la 'l'ième ligne dans la matrice de 'tab'.

appelle : calculvinc, lire, iemedef, ecrire

est appelée par : simulateur

Procédure **ECRIREPI** (chaine, tab, l)

argum. : chaîne : vectstring
vectncol : vectcol (variable globale)
p : vectval (variable globale)
l : entier
tab : tabres

précond. : . la matrice de 'tab' n'est pas vide
. 'l' est compris entre 1 et le nombre (logique) de lignes de la matrice 'tab'

. 'chaîne' n'est pas vide
. 'chaîne' comprend autant d'éléments que le vecteur-définition de 'tab', et les noms repris dans 'chaîne' sont identiques aux noms des variables définies dans 'tab' (ordre non significatif)
. les valeurs de toutes les variables ont étés calculées et inscrites dans la matrice de 'tab' pour les lignes 1 à 'l'-1
. le vecteur 'p' n'est pas vide et comprend autant d'éléments que le vecteur 'chaîne' (les longueurs logiques de 'p' et de 'chaine'sont identiques)
. le ième élément du vecteur 'vectncol' est égal à la position (c'est-à-dire au numero d'ordre) dans le vecteur-définition de 'tab' de la définition dont le nom est inscrit en ième position du vecteur 'chaine'

spécif. : pour chaque ième élément (soit A) du vecteur 'chaîne' : si A est le nom d'une variable définie dans 'tab' comme étant une variable dépendante ou évolutive, écrire la ième valeur du vecteur 'p' à la 'l'ième ligne et à la colonne correspondant à cette variable dans la matrice de 'tab'

appelle : iemedef, ecrire

est appelée par : simulateur

Procédure **CREERPAS** (tab, b1, b2, pas)

argum. : tab: tabres
b1, b2, pas : réels

résul. : tab : tabres
message d'erreur

spécif. : si la matrice de 'tab' n'est pas vide, afficher un message d'erreur,
sinon attribuer à 'tab' une matrice à une colonne et contenant tous les nombres générés par pogression donnée (valeur absolue de 'pas') à partir d'une borne inférieure ('b1' si 'b1'<='b2','b2' sinon, (limite comprise)), et ne dépassant pas une borne supérieure ('b1' si 'b1'> 'b2','b2' sinon). Il y a autant de lignes dans la matrice de 'tab' que de nombres ainsi générés

appelle : creermat, ecrire, exist

Procédure **JUXTAVAL** (tab1, tab2, tab3)

argum. : tab1, tab2, tab3 : tabres

précond. : . les matrices de 'tab1' et 'tab2' ne sont pas vides et contiennent le même nombre de lignes
. la matrice de 'tab3' a autant de lignes que la matrice de 'tab1' et autant de colonnes que la somme des colonnes des matrices de 'tab1' et 'tab2'

résul. : tab3 : tabres

spécif. : recopier en respectant l'ordre des lignes et des colonnes toutes les colonnes de valeurs de la matrice de 'tab1' (soit i colonnes) dans les i premières colonnes de la matrice de 'tab3', et toutes les colonnes de valeurs de la matrice de 'tab2' (soit j colonnes) dans les j dernières colonnes de 'tab3' (càd à la droite de la copie, dans 'tab3', de la matrice de 'tab1').

appelle : lgmat, lrmat, lire, ecrire

est appelée par : juxtamat, juxtatab

Procédure **JUXTAMAT** (tab1, tab2, tab3)

argum. : tab1, tab2, tab3 : tabres

résul. : tab3 : tabres
messages d'erreur

spécif. : si la matrice de 'tab1' ou 'tab2' est vide,
ou si la matrice de 'tab3' n'est pas vide,
ou si les matrices de 'tab1'et 'tab2' ne contiennent pas le même nombre de lignes,
ou s'il est impossible d'attribuer à 'tab3' une matrice contenant autant de lignes que la matrice de 'tab1' et autant de colonnes que la somme des colonnes des matrices de 'tab1' et 'tab2',
afficher un message d'erreur,
sinon . attribuer à 'tab3' une matrice contenant autant de lignes que la matrice de 'tab1' et autant de colonnes que la somme des colonnes des matrices de 'tab1' et 'tab2',
. et recopier en respectant l'ordre des lignes et des colonnes toutes les colonnes de valeurs de la matrice de 'tab1' (soit i

colonnes) dans les i premières colonnes de la matrice de 'tab3', et toutes les colonnes de valeurs de la matrice de 'tab2' (soit j colonnes) dans les j dernières colonnes de 'tab3' (càd à la droite de la copie, dans 'tab3', de la matrice de 'tab1')

appelle : juxtaval, exist, creermat, lgmat, lrmat

Procédure **JUXTADEF** (tab1, tab2, tab3)

argum. : tab1, tab2, tab3 : tabres

précond. : . les vecteurs-définition de 'tab1' et 'tab2' ne sont pas vides
. 'tab3' peut au moins contenir un nombre de définitions de variable égal à la somme des définitions de 'tab1' et 'tab2'
. aucune variable définie dans 'tab1' ne porte le même nom qu'une variable définie dans 'tab2'

résul. : tab3 : tabres

spécif. : . si le vecteur-définition de 'tab3'n'est pas vide, le mettre à vide;
. recopier, en respectant l'ordre, toutes les définitions de variable du vecteur définition de 'tab1' (soit i définitions) dans les i premiers éléments du vecteur-définition de 'tab3', et toutes les définitions du vecteur-définition de 'tab2' (soit j définitions) dans les j éléments suivants du vecteur-définition de 'tab3'.

appelle : vectvide, initvect, vectlg, iemedef, ecriredef

est appelée par : juxtavect, juxtatab, prodtab

Procédure **JUXTAVECT** (tab1, tab2, tab3)

argum. : tab1, tab2, tab3 : tabres

résul. : tab3 : tabres
messages d'erreur

spécif. : si le vecteur-définition de 'tab1' ou 'tab2' est vide,
ou si 'tab3' ne peut contenir un nombre de définitions de variables égal à la somme des

définitions des vecteurs-définition de 'tab1' et 'tab2',
ou si (au moins) une variable définie dans 'tab1' porte le même nom qu'une variable définie dans 'tab2',
alors afficher un message d'erreur
sinon .si le vecteur-définition de 'tab3'n'est pas vide, le mettre à vide;
.recopier, en respectant l'ordre, toutes les définitions de variables du vecteur définition de 'tab1' (soit i définitions) dans les i premiers éléments du vecteur-définition de 'tab3', et toutes les définitions du vecteur-définition de 'tab2' (soit j définitions) dans les j éléments suivants du vecteur-définition de 'tab3'.

appelle : juxtadef, vectvide, veriflgvect, vectlg, memenom

Procédure **JUXTATAB** (tab1, tab2, tab3)

argum. : tab1, tab2, tab3 : tabres

résul. : tab3 : tabres
messages d'erreur

spécif. : si le tableau 'tab1' ou 'tab2' n'est pas cohérent (voir procédure coherence),
ou si la matrice de 'tab3' n'est pas vide
ou si les matrices de 'tab1' et 'tab2' n'ont pas le même nombre de lignes,
ou si 'tab3' ne peut contenir un nombre de définitions de variable égal à la somme des définitions des vecteurs-définition de 'tab1' et 'tab2',
ou si (au moins) une variable définie dans 'tab1' porte le même nom qu'une variable définie dans 'tab2',
ou s'il est impossible d'attribuer à 'tab3' une matrice contenant autant de lignes que la matrice de 'tab1' et autant de colonnes que la somme des colonnes des matrices de 'tab1' et 'tab2',
afficher un message d'erreur,
sinon -attribuer à 'tab3' une matrice contenant autant de lignes que la matrice de 'tab1' et autant de colonnes que la somme des colonnes des matrices de 'tab1' et 'tab2',
-et recopier en respectant l'ordre des lignes et des colonnes toutes les

colonnes de valeurs de la matrice de 'tab1' (soit i colonnes) dans les i premières colonnes de la matrice de 'tab3', et toutes les colonnes de valeurs de la matrice de 'tab2' (soit j colonnes) dans les j dernières colonnes de 'tab3' (càd à la droite de la copie, dans 'tab3', de la matrice de 'tab1') et
-si le vecteur-définition de 'tab3'n'est pas vide, le mettre à vide, et
-recopier, en respectant l'ordre, toutes les définitions de variables du vecteur-définition de 'tab1' (soit i définitions) dans les i premiers éléments du vecteur-définition de 'tab3', et toutes les définitions du vecteur-définition de 'tab2' (soit j définitions) dans les j éléments suivants du vecteur-définition de 'tab3'.

appelle : coherence, exist, lgmat, lrmat, veriflgvect, memenom, creermat, juxtaval, juxtadef, vectlg

Procédure **COPIERVAL** (tab1, tab2)

argum. : tab1, tab2 : tabres

précond. : . la matrice de 'tab1' n'est pas vide
. la matrice de 'tab2' contient autant de lignes et autant de colonnes que la matrice de 'tab2'

résul. : tab2 : tabres

spécif. : copier dans la matrice de 'tab2' les valeurs reprises dans la matrice de 'tab1' en respectant l'ordre des lignes et des colonnes

appelle : lgmat, lrmat, ecrire, lire

est appelée par : copiermat, copiertab

Procédure **COPIERMAT** (tab1, tab2)

argum. : tab1, tab2 : tabres

résul. : tab2 : tabres
messages d'erreur

spécif. : si la matrice de 'tab1' est vide,
ou si la matrice de 'tab2' n'est pas vide,
ou s'il est impossible d'attribuer à 'tab2' une matrice contenant autant de lignes et de colonnes que la matrice de 'tab2',
alors afficher un message d'erreur
sinon -attribuer à 'tab2' une matrice contenant autant de lignes et de colonnes que la matrice de 'tab1', et
-copier dans la matrice de 'tab2' les valeurs reprises dans la matrice de 'tab1' en respectant l'ordre des lignes et des colonnes

appelle : exist, creermat, copierval, lgmat, lrmat

Procédure **COPIERDEF** (tab1, tab2)

argum. : tab1, tab2 : tabres

précond. : . le vecteur-définition de 'tab1' n'est pas vide
. il est possible d'attribuer à 'tab2' un vecteur-définition de longueur égale à celui de 'tab1'

résul. : tab2 : tabres

spécif. : . si le vecteur-définition de 'tab2' n'est pas vide, le mettre à vide
. copier dans le même ordre, les définitions de variable du vecteur-définition de 'tab1' dans le vecteur-définition de 'tab2'

appelle : vectlg, vectvide, initvect, iemedef, ecriredef

est appelé par : copiervect, copiertab

Procédure **COPIERVECT** (tab1, tab2)

argum. : tab1, tab2 : tabres

résul. : tab2 : tabres
messages d'erreur

spécif. : si le vecteur-définition de 'tab1' est vide,
ou s'il n'est pas possible d'attribuer à 'tab2' un vecteur-définition de longueur égale à celui de 'tab1', afficher un message d'erreur,
sinon . si le vecteur-définition de 'tab2' n'est pas vide, le mettre à vide

. copier dans le même ordre, les définitions de variable du vecteur-définition de 'tab1' dans le vecteur-définition de 'tab2'

appelle : vectvide, veriflgvect, vectlg, copierdef

Procédure COPIERTAB (tab1, tab2)

argum. : tab1, tab2 : tabres

résul. : tab2 : tabres
messages d'erreur

spécif. : si la matrice ou le vecteur-définition de 'tab1' est vide,
ou si la matrice de 'tab2' n'est pas vide,
ou s'il n'est pas possible d'attribuer à 'tab2' un vecteur-définition de longueur égale à celui de 'tab1',
ou s'il n'est pas possible d'attribuer à 'tab2' une matrice contenant autant de lignes et de colonnes que la matrice de 'tab1',
alors afficher un message d'erreur,
sinon . copier dans la matrice de 'tab2' les valeurs reprises dans la matrice de 'tab1' en respectant l'ordre des lignes et des colonnes et
. si le vecteur-définition de 'tab2' n'est pas vide, le mettre à vide ,
. et copier dans le même ordre, les définitions de variable du vecteur-définition de 'tab1' dans le vecteur-définition de 'tab2'

appelle : exist, vectvide, veriflgvect, creermat, copierval, copierdef, vectlg, lgmat, lrmat

Procédure PRODVAL (tab1, tab2)

argum. : tab1, tab2, tab3 : tabres

précond. : . les matrices de 'tab1' et de 'tab2' ne sont pas vides
. la matrice de 'tab3' à autant de colonnes que la somme des colonnes des matrices de 'tab1' et 'tab2' et autant de lignes que le produit du nombre de lignes de la matrice de 'tab1' par le nombre de lignes de la matrice de 'tab2'

résul. : tab3 : tabres

spécif. : soient i et p respectivement égaux au nombre de lignes et au nombre de colonnes de la matrice de 'tab1', et j et q respectivement égaux au nombre de lignes et au nombre de colonnes de la matrice de 'tab2'.
Remplir la matrice de 'tab3' en effectuant le produit cartésien ordonné de la matrice de 'tab1' par la matrice de 'tab2' :
- les p premières colonnes de la matrice de 'tab3' correspondent aux p colonnes de la matrice de 'tab1', et les q dernières colonnes de la matrice de 'tab3' correspondent aux q colonnes de la matrice de 'tab2',
- chaque ligne de la matrice de 'tab1' ('tab2') est recopiée j fois (i fois) dans la matrice de 'tab3'
(pour les schémas, revoir les concepts relatifs aux manipulations de matrices : produit cartésien ordonné de deux matrices)

appelle : lgmat, lrmat, lire, ecrire

est appelée par : prodmat, prodtab

Procédure **PRODMAT** (tab1, tab2)

argum. : tab1, tab2, tab3 : tabres

résul. : tab3 : tabres
messages d'erreur

spécif. : si la matrice de 'tab1' ou de 'tab2' est vide, ou si la matrice de 'tab3' n'est pas vide, ou s'il est impossible d'affecter à 'tab3' une matrice contenant autant de lignes que le produit du nombre de lignes de la matrice de 'tab1' par le nombre de lignes de la matrice de 'tab2', et autant de colonnes que la somme des colonnes des matrices de 'tab1' et de 'tab2', alors afficher un message d'erreur ,
sinon remplir la matrice de 'tab3' en effectuant le produit cartésien ordonné de la matrice de 'tab1' par la matrice de 'tab2'.

appelle : exist, creermat, prodval, lgmat, lrmat, prodval

Procédure **PRODTAB** (tab1, tab2, tab3)

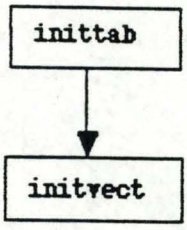
argum. : tab1, tab2, tab3 : tabres

résul. : tab2 : tabres
messages d'erreur

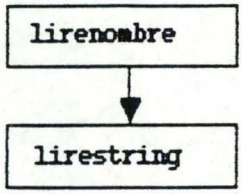
spécif. : si le tableau 'tab1' ou 'tab2' n'est pas cohérent (voir procédure coherence),
ou si la matrice de 'tab3'b'est pas vide
ou si les matrices de 'tab1' et 'tab2' n'ont pas le même nombre de lignes,
ou si 'tab3' ne peut contenir un nombre de définitions de variables égal à la somme des définitions des vecteurs-définition de 'tab1' et 'tab2',
ou si (au moins) une variable définie dans 'tab1' porte le même nom qu'une variable définie dans 'tab2',
ou s'il est impossible d'affecter à 'tab3' une matrice contenant autant de lignes que le produit du nombre de lignes de la matrice de 'tab1' par le nombre de lignes de la matrice de 'tab2', et autant de colonnes que la somme des colonnes des matrices de 'tab1' et de 'tab2',
alors afficher un message d'erreur
sinon . remplir la matrice de 'tab3' en effectuant le produit cartésien ordonné de la matrice de 'tab1' par la matrice de 'tab2' et
. si le vecteur-définition de 'tab3'n'est pas vide, le mettre à vide,
. recopier, en respectant l'ordre, toutes les définitions de variables du vecteur définition de 'tab1' (soit i définitions) dans les i premiers éléments du vecteur-définition de 'tab3', et toutes les définitions du vecteur-définition de 'tab2' (soit j définitions) dans les j éléments suivants du vecteur-définition de 'tab3'.

appelle : exist, verif1gvect, vectlg, memenom, prodval, juxtadef, coherence, creermat, lgmat, lrmat

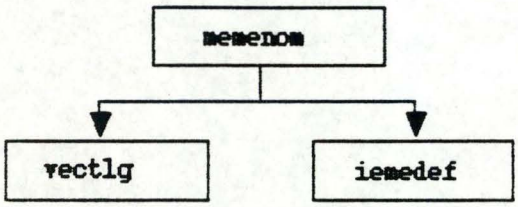
ANNEXE 2 : ARCHITECTURE PHYSIQUE



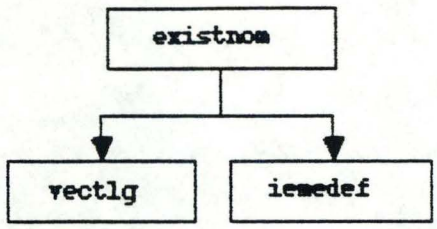
Architecture physique de inittab



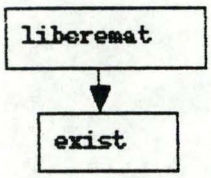
Architecture physique de lirenombre



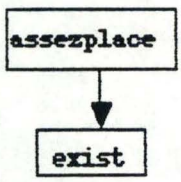
Architecture physique de memenom



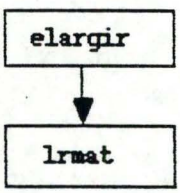
Architecture physique de existnom



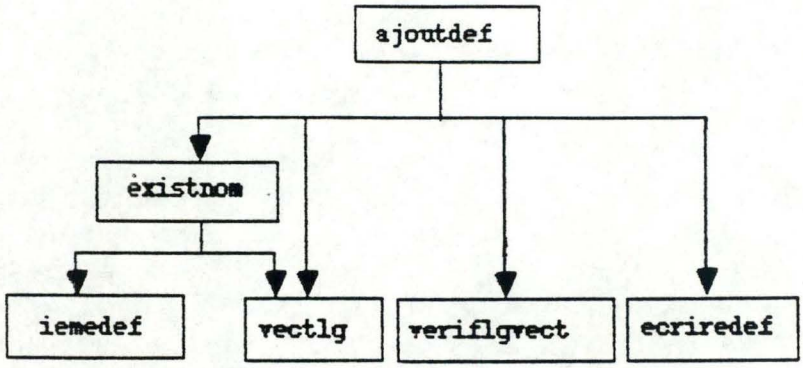
Architecture physique de liberemat



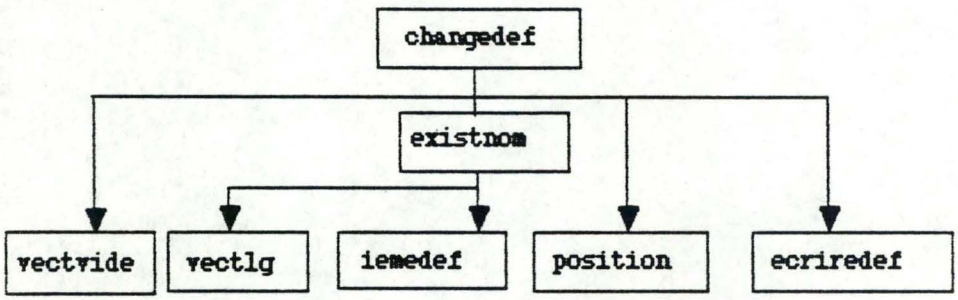
Architecture physique de assezplace



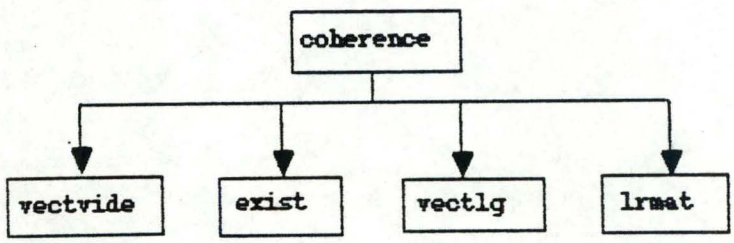
Architecture physique de elargir



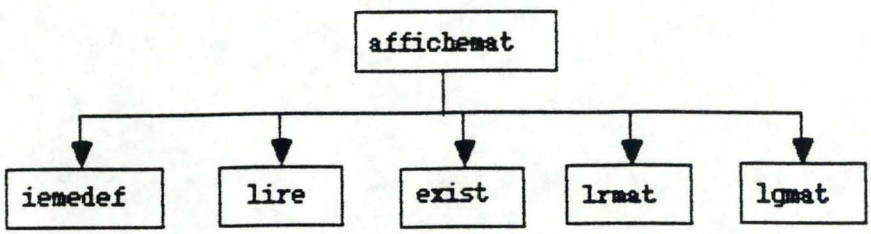
Architecture physique de ajoutdef



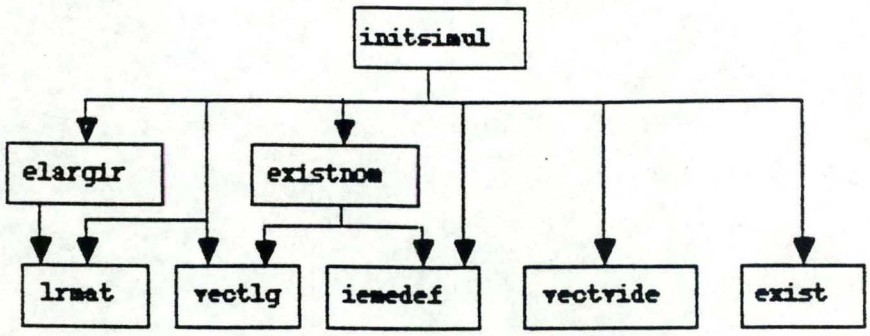
Architecture physique de changedef



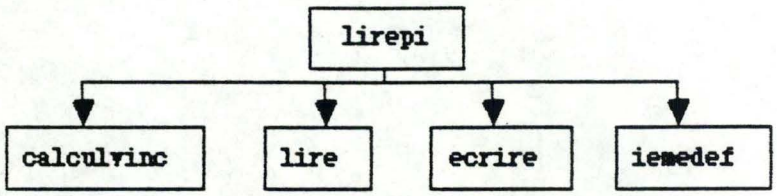
Architecture physique de coherence



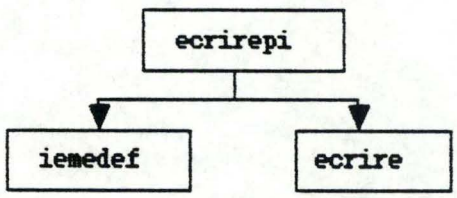
Architecture physique de affichemat



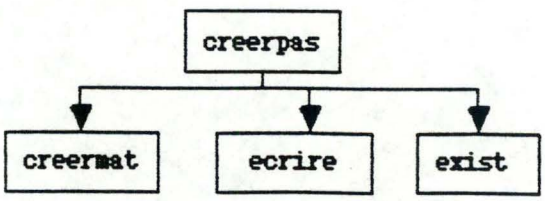
Architecture physique de initsimul



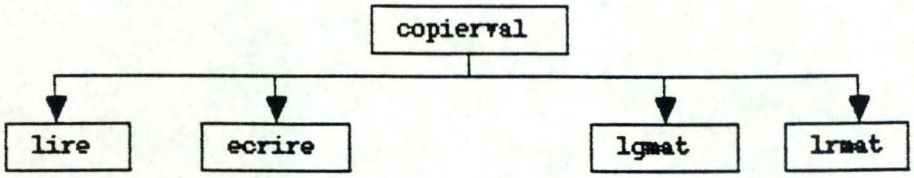
Architecture physique de lirepi



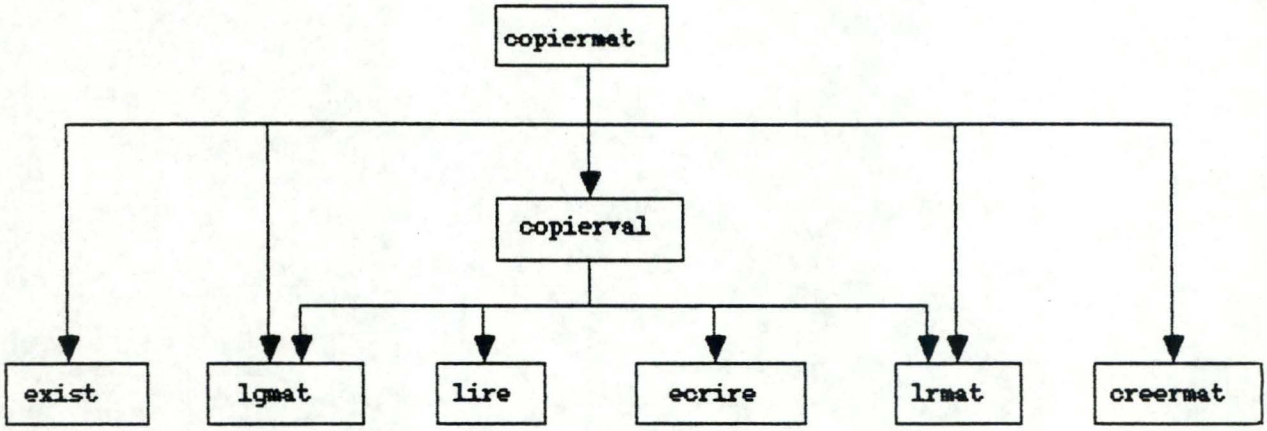
Architecture physique de ecrirepi



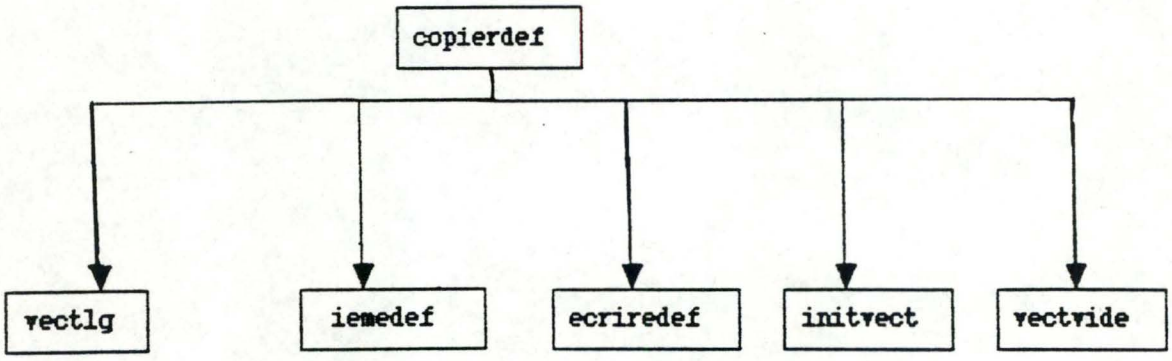
Architecture physique de creerpas



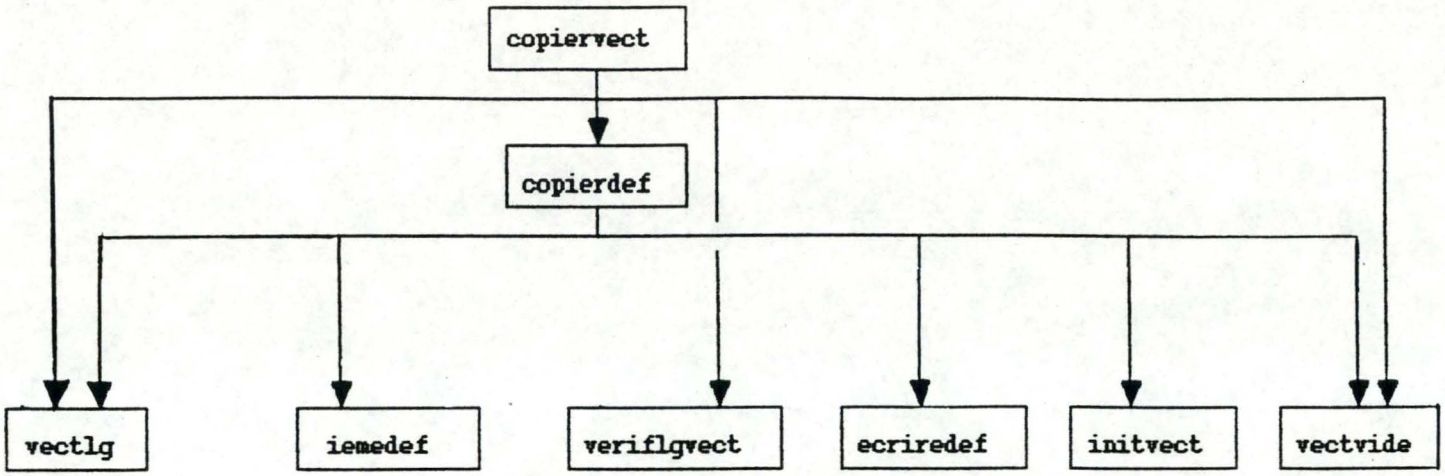
Architecture physique de copierval



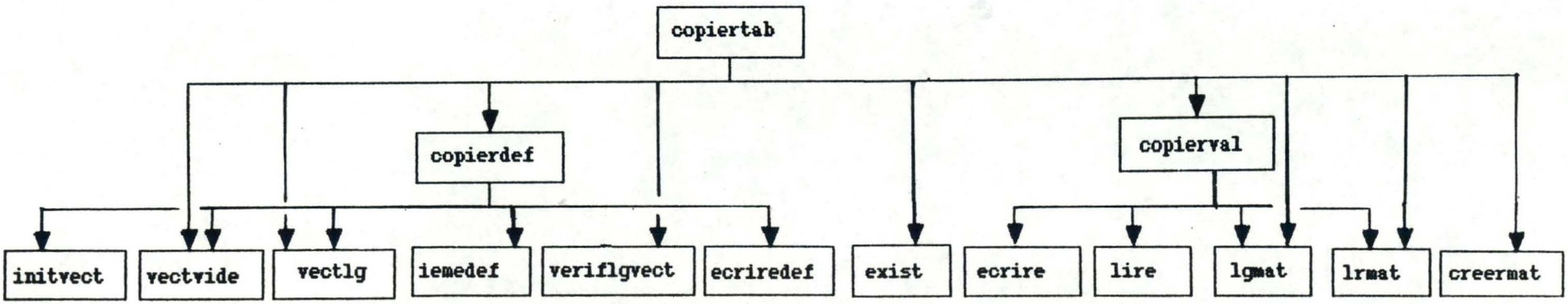
Architecture physique de copiermat



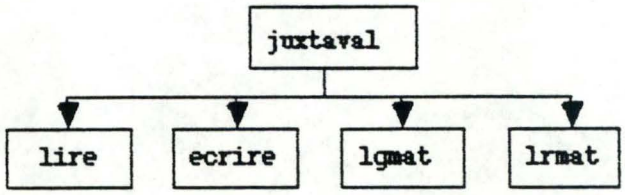
Architecture physique de copierdef



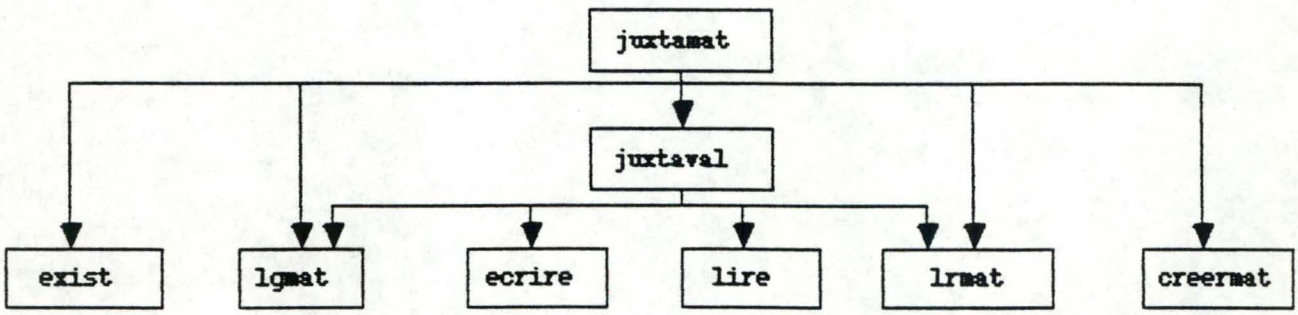
Architecture physique de copiervect



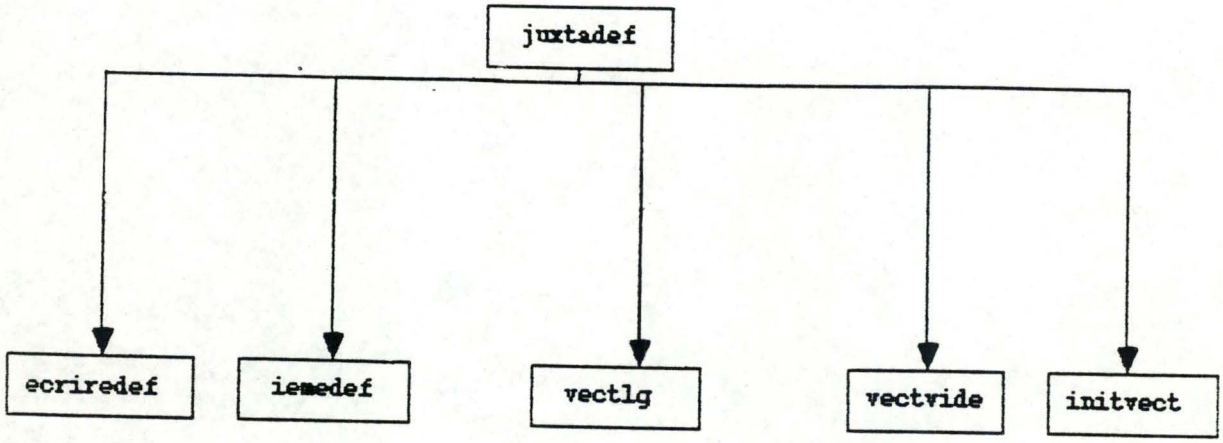
Architecture physique de copiertab



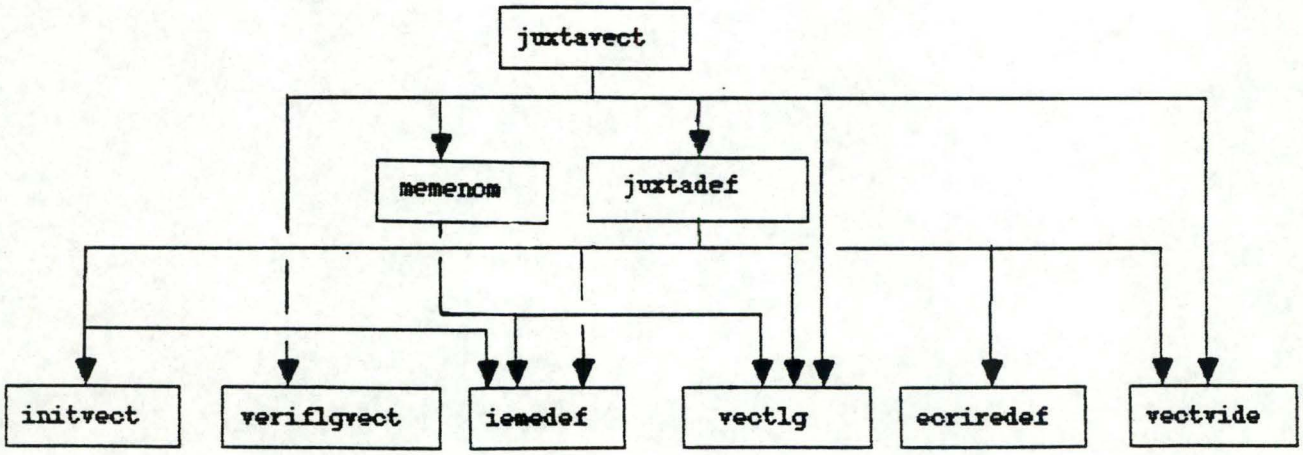
Architecture physique de juxtaval



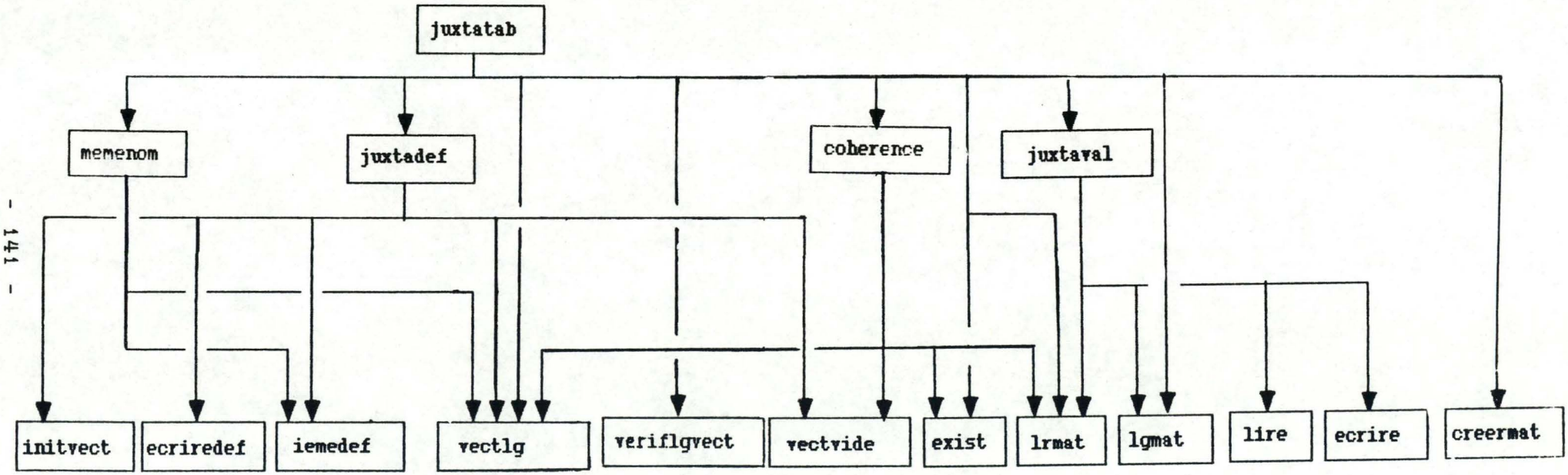
Architecture physique de juxtamat



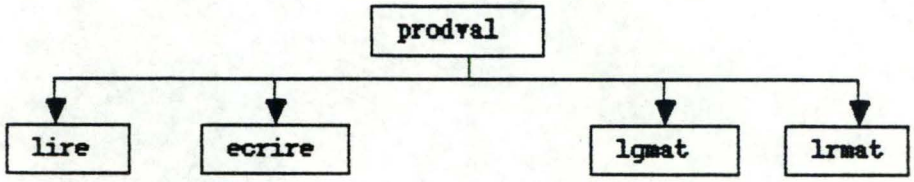
Architecture physique de juxtadef



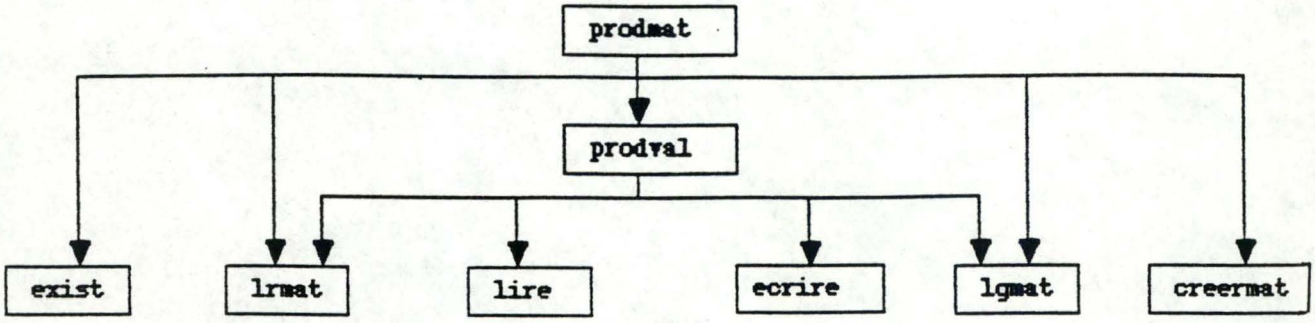
Architecture physique de juxtavect



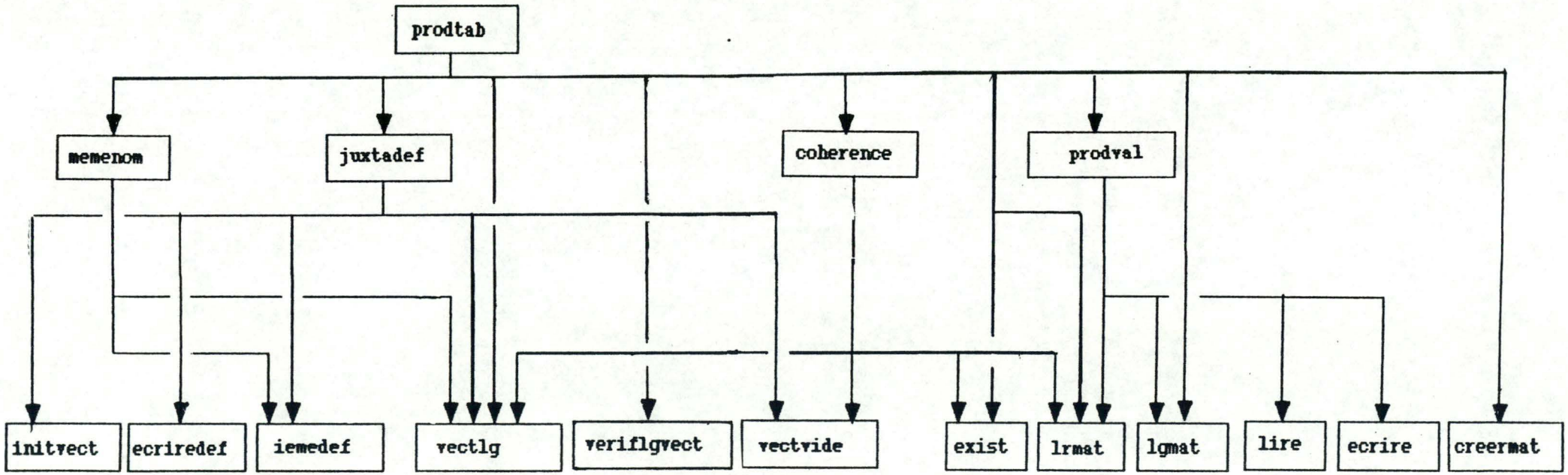
Architecture physique de juxtatab



Architecture physique de prodval



Architecture physique de prodmat



Architecture physique de prodtab

ANNEXE 3 : TEXTE DU SYSTEME

```
(*$r+*)
const longmax = 500;
      largmax = 20;
      longstring = 50;
      maxcolecran = 80;
      maxlignecran = 25;



type pttab = ^tabres;
     lien = ^objet;
     objet = record data : integer;
                    tab : pttab;
                    suivt : lien end;
     genrevar = (vi, vd, ve);
     modet = (con, nonc);
     strnm = string[8];
     strut = string[10];
     stringmax = string[longstring];
     variable = record nom : strnm;
                       unite : strut;
                       case genre : genrevar of
                              vd : ();
                              vi : ( case md : modet of
                                       con : ();
                                       nonc : () );
                            ve : (origine : real) end;
     tabres = record vect : array[1..largmax] of variable;
                     lg : integer;
                     lr : integer;
                     pt : lien end;
     vectstring = array[1..largmax] of strnm;
     setchar = set of char;
     vectval = array [1..largmax] of real;
     vectcol = array [1..largmax] of integer;



var  tabgen : record tabval : array[1..longmax,1..largmax] of real;
                     premier : lien;
                     libre : integer end;
     p: vectval;
     vectncol: vectcol;
```

```
procedure lirestring ( var strg: stringmax; x,y,long : integer;
                       okset: setchar);

   var a,b,i:integer;
       ch:char;


   procedure lirech (var ch:char; okset : setchar);

    begin
     read (kbd, ch);
     if eoln(kbd) then ch := chr (13);
     while not (ch in okset) do
       begin
       write(chr(7));
       read (kbd,ch);
       if eoln(kbd) then ch:=chr(13)
       end
    end;


   procedure points (x,y,long:integer);

    var i : integer;

    begin
     gotoxy(x,y);
     for i := 1 to long do
       begin
       write('.');
       end;
     gotoxy(x,y);
    end;


begin
 b := long -(maxcolecran - x) -1;
 a := (b div maxcolecran) + 1;
 if (x < 1) or (x > maxcolecran) or (y < 1) or (y > maxlignecran)
   then begin
        writeln ('erreur, les coordonnées ecran sont invalides
    (lirestring)');
        readln
        end
   else  if long > longstring
           then begin
                writeln   ('erreur,   le   nombre   de   points   a
afficher est supérieur');
                writeln ('à la longueur déclarée du string
(lirestring)');
                readln
                end
            else  if y + A > maxlignecran
                  then begin
```

```
                        writeln ('erreur, tous les points à afficher
ne tombent pas');
                        writeln ('dans les limites de l''écran
(lirestring)');
                        readln
                        end
                    else
   begin
     points(x,y,long);
     strg:='';
     i:=0;
     repeat
       lirech (ch, okset + [chr(13),chr(8)]);
       if ch in okset
         then begin
              if i <> long
                then begin
                     i := i + 1;
                     strg := concat (strg,' ');
                     strg[i] := ch;
                     write (ch)
                     end
                else write (chr(7));
              end
         else if ch = chr (8)
               then if i <> 0
                     then begin
                          delete (strg, length(strg), 1);
                          i := i - 1;
                          points (x + i, y , 1);
                          end
                     else write (chr(7));
       until ch = chr(13);
       for i := length(strg) + 1 to long do write(' ')
     end
    end;



procedure lirenombre (var r:real; x,y: integer);

 var st:stringmax;
     code,i :integer;


 begin
   if (x < 1) or (x > maxcolecran) or (y < 1) or (y > maxlignecran)
     then begin
          writeln ('erreur, les coordonnées ecran sont invalides
(lirenombre)');
          readln
          end
     else begin
   code := 3;
   while code <> 0 do
```

```
   begin
   lirestring (st,x,y,16,['0'..'9','-','+','E','.']);

   val(st,r,code);
   if code <> 0 then begin
                     gotoxy(1,20);
                     writeln ('erreur, ceci n''est pas un réel
acceptable, recommencez');
                     end;
   end;
   gotoxy (1,20);
   for i := 1 to 55 do
   write(' ')
end
end;



procedure lire (var tab: tabres; col, lig: integer; var res:real);

  var debut,ligne: integer;

  begin
     debut := tab.pt^.data;
     ligne := debut + lig - 1;
     res := tabgen.tabval[ligne,col]
  end;



procedure ecrire (var tab: tabres; col, lig : integer; val :real);


  var debut, ligne : integer;

  begin
     debut := tab.pt^.data;
     ligne := debut + lig - 1;
     tabgen.tabval[ligne, col] := val

  end;



function lgmat (tab:tabres): integer;

   begin
     lgmat := tab.lg
   end;
```

```
function lrmat (tab:tabres): integer;

    begin
      lrmat := tab.lr
    end;



function vectlg (tab:tabres): integer;

    var i : integer;
        ok: boolean;
    begin
      i := 1;
      ok := true;
      while (i <> largmax + 1) and ok do
          if tab.vect[i].nom <> '' then
            i := i + 1
                                      else
           ok := false;
      vectlg := i- 1;
    end;



procedure iemedef ( tab: tabres; var def:variable; i:integer);

    begin
      def := tab.vect[i]
    end;



function vectvide(tab:tabres): boolean;

  begin
   if tab.vect[1].nom <> '' then vectvide := false
                            else vectvide := true
 end;



procedure initvect (var tab: tabres);

   var i : integer;

   begin
       for i := 1 to largmax do
           begin
```

```
            tab.vect[i].nom := '';
            tab.vect[i].unite := ''
            end;
   end;



procedure inittab (var tab:tabres);

   begin
     initvect (tab);
     tab.lr := 0;
     tab.lg := 0
   end;



function veriflgvect (tab:tabres; i:integer): boolean;

    begin
      if i  > largmax then veriflgvect := false
                      else veriflgvect := true;
    end;



function mememom (tab1, tab2 : tabres): boolean;

   var i,j : integer;
       def1, def2: variable;
       ok : boolean;


   begin
     i:= 1; ok:= false;
     while (i <= vectlg(tab1)) and not ok do
        begin
        iemedef (tab1, def1, i);
        j := 1;
        while (j <= vectlg(tab2)) and not ok do
          begin
          iemedef (tab2,def2,j);
          if def1.nom = def2.nom then ok := true;
          j := j + 1;
          end;
        i := i + 1;
        end;
     mememom := ok;
   end;
```

```
function existnom (tab:tabres; nom:strnm ): boolean;

      var i :integer;
          vari : variable;
          ok : boolean;

    begin
      i := 1;
      ok := false;
      while (i <= vectlg (tab)) and not ok do
          begin
            iemedef (tab, vari, i);
            if nom = vari.nom then ok := true;
            i := i + 1;
          end;
      existnom := ok;
    end;



procedure defevol (var def: variable; nom : strnm; unite : strut;
                   orig: real);

  begin
    if nom = '' then begin
                     writeln ('erreur, le nom donné est une
 chaîne de caractères vide (defevol)');
                     readln
                     end
                 else begin
    def.nom := nom;
    def.unite := unite;
    def.genre := ve;
    def.origine := orig  end;
  end;



procedure defdep (var def : variable; nom : strnm; unite : strut);

  begin
    if nom = '' then begin
                     writeln ('erreur, le nom donné est une chaîne
de caractères vide (defdep)');
                     readln
                     end
                 else begin
    def.nom := nom;
    def.genre := vd;
    def.unite := unite
    end;
  end;
```

```
procedure deficon (var def : variable; nom : strnm; unite : strut);

  begin
    if nom = '' then begin
                     writeln ('erreur, le nom donné est une chaîne
de caractères vide (deficon)');
                     readln
                     end
                else begin
    def.nom := nom;
    def.unite := unite;
    def.genre := vi;
    def.md := con
        end;
  end;



procedure ecriredef (var tab: tabres; def: variable; i:integer);

  begin
    tab.vect[i] := def
  end;



function position (tab:tabres; nom :strnm): integer;


  var i:integer;
      ok: boolean;

 begin
   i := 1; ok:= false;
   while (i<= largmax) and not ok do
     begin
     if tab.vect[i].nom = nom then ok := true;
     i := i+ 1;
     end;
     if not ok then position := 0
               else position := i - 1;
   end;



 procedure ajoutdef (var tab:tabres; def: variable);

   begin
     if not veriflgvect (tab, vectlg (tab) + 1)
```

```
        then begin
             writeln ('erreur, le futur vecteur serait trop long
(ajoutdef)');
             readln
             end
         else if existnom (tab, def.nom)
              then begin
                   writeln  ('erreur,  le nouveau  nom  existe
déjà (ajoutdef)');
                   readln
                   end
               else ecriredef (tab, def, vectlg(tab)+1);
   end;



procedure changedef (var tab:tabres; nom: strnm; def: variable);


  begin
    if vectvide (tab)
      then begin
           writeln ('erreur, le vecteur-définition original est
vide (changedef)');
           readln
           end
       else if nom =''
            then begin
                 writeln ('erreur, le nom donné est une chaîne
 de caractères vide (changedef)');
                 readln
                 end
            else if not existnom (tab, nom)
                 then begin
                      writeln ('erreur, le nom donné n''est
pas un nom de variable');
                      writeln ('appartenant au vecteur-
définition (changedef)');
                      readln
                      end
                 else if (existnom (tab,def.nom)) and
                         ( nom <> def.nom)
                   then begin
                                writeln ('erreur, le nouveau
 vecteur-définition comprendrait deux variables ');
                                writeln ('de même nom (changedef)');
                                readln
                        end
                   else ecriredef(tab, def, position(tab,nom));
  end;
```

```
function exist (var tab :tabres) : boolean;

 var lg : integer;
     trouve : boolean;
     cour : lien;

 begin
   cour := tabgen.premier;
   trouve := false;
   if cour^.tab = addr(tab) then trouve := true;
   while (trouve = false) and (cour^.suivt <> nil) do
      begin
        cour := cour^.suivt;
        if ((seg(cour^.tab^.vect[1]) = seg (tab))
           and (ofs(cour^.tab^.vect[1]) = ofs (tab)))
                    then trouve := true;
      end;

    if trouve then exist := true
           else exist := false;
 end;



procedure affichemat ( var tab:tabres);

   var   x:char;
         ori, debut, mini, minj, maxi, maxj, i, j:integer;
         def:variable;
         r:real;

  procedure repete;
   begin
    ori := 12;
    for j := minj to maxj do begin
        gotoxy (ori,1);
        iemedef(tab,def,j);
        write (def.nom);
        gotoxy (ori,2);
        if def.unite <> '' then
        write ('(',def.unite,')');
        ori := ori + 18;
        end;


     ori := 4;
    for i:= mini to maxi do begin
       gotoxy(1,ori);
       write(i);
       gotoxy(6,ori);
       for j:= minj to maxj do
          begin
          lire (tab,j,i,r);
```

```
          write (r);
          end;
          writeln;
     ori := ori + 1;

                                   end;
  end;



begin
   clrscr;
   if not exist(tab) then begin
                          writeln ('erreur, la matrice est vide
(affichemat)');
                          readln
                          end
                     else begin
   if 4 > lrmat(tab) then maxj := lrmat(tab) else maxj := 4;
   if 20 > lgmat(tab) then maxi := lgmat(tab) else maxi := 20;

   mini := 1;
   minj := 1;
   repete;

   reset(kbd);
   read (kbd,x);
   while ord(x) <> 13 do
      begin
      if (keypressed)
       then begin
            read (kbd,x);
            if ord(x) = 80
        then begin
             if maxi <> lgmat(tab)
                then begin
                     clrscr;
                     mini := maxi + 1;
                     if maxi + 20 > lgmat(tab)
                            then maxi := lgmat(tab)
                            else maxi := maxi + 20;
                     repete;
                     end;
               end
         else if ord(x) = 72
                then begin
                     if mini <> 1
                        then begin
                             clrscr;
                             maxi := mini - 1;
                             mini := mini- 20;
                             repete;
                             end;
                    end
                else if ord(x) = 77
                       then begin
```

```
                                    if maxj <> lrmat(tab)
                                      then begin
                                           clrscr;
                                           minj := maxj + 1;
                                           if maxj + 4 > lrmat(tab)
                                              then maxj := lrmat(tab)
                                              else maxj := maxj + 4;
                                           repete;
                                           end;
                                     end
                                else if ord(x) = 75
                                       then begin
                                            if minj <> 1
                                               then begin
                                                    clrscr;
                                                    maxj := minj - 1;
                                                    minj := minj - 4;
                                                    repete
                                                    end;
                                             end;
                        end;
              reset(kbd);
              read(kbd,x)
                end;
            clrscr
            end
            end;



procedure initialisation;

begin
  tabgen.tabval[1,1] := longmax;
  tabgen.tabval[1,2] := 0;
  tabgen.premier := nil;
  tabgen.libre := 1;
end;



procedure ajoutnom (var chaine:vectstring; nom : strnm);

  var i: integer;
      ok: boolean;


  function dejanom: boolean;

    var i:integer;
        ok: boolean;

   begin
     i := 1;
```

```
      ok := false;
      while (i <= largmax) and not ok do


        begin
         if chaine[i] = nom then ok := true;
         i := i + 1
        end;
      dejanom := ok
    end;


  begin
   i := 1;
   ok := true;
   while (i <> largmax + 1) and ok do
   if chaine[i] <> '' then i := i + 1
                      else ok := false;
   if nom = ''
     then begin
          writeln ('erreur, le nom donné est une chaîne de
caractères vide (ajoutnom)');
          readln
          end
     else if i >= largmax + 1
           then begin
                writeln ('erreur, il n''y a plus de place dans le
 vecteur de noms (ajoutnom)');
                readln
                end
           else if dejanom
                   then begin
                        writeln ('erreur, le nom donné existe
 déjà dans le vecteur de noms (ajoutnom)');
                        readln
                        end
            else chaine[i] := nom;
  end;



procedure initchaine (var chaine: vectstring);

  var i: integer;

  begin
   for i := 1 to largmax do
     chaine[i] := ''
  end;
```

```
function coherence( var tab :tabres): boolean;

  begin
    if not vectvide(tab) and exist(tab) and
          (vectlg(tab) = lrmat(tab))
        then  coherence := true
        else  coherence := false
   end;



procedure elargir (var tab:tabres; i:integer; var bon:boolean);

  begin
     bon:= true;
     tab.lr := lrmat(tab) + i
  end;



procedure initsimul (chaine: vectstring; var t:tabres;
                     var ok:boolean ; x:integer);


  var i,j: integer;
      nom : strnm;
      def: variable;
      bon: boolean;


   function egalg: boolean;

     var i:integer;
         ok: boolean;

     begin
      i:= 1;
      ok := true;
      while (i<> largmax + 1) and ok do
         if chaine[i] <> '' then i:= i + 1
                            else ok := false;
      i := i-1;
      if i <> vectlg (t) then egalg := false
                         else egalg := true;
     end;


   function memevect: boolean;

     var i:integer;
         ok : boolean;
```

```
    begin
      i := 1;
      ok := true;
      while (i<= vectlg(t)) and ok do
        if not existnom (t,chaine[i]) then ok := false
                                      else i := i + 1;
      mememvect := ok
    end;


  procedure okvi ( var nom :strnm; var ok: boolean);

    var i: integer;
        def: variable;

    begin
      i := 1;
      ok := true;
      while (i <= vectlg(t)) and ok do
        begin
        iemedef(t,def,i);
        if (def.genre = vi) and (def.md = con) and (lrmat(t) < i)
           then ok := false
           else i := i + 1;
        end;
      nom := def.nom
    end;



  begin

   ok := false;
   if not exist(t)
     then begin
          writeln ('erreur, la matrice est vide (simulateur)');
          readln
          end
     else
   if (x < 1) or (x > largmax)
     then begin
          writeln ('erreur, le nombre de paramètres donnés par
le constructeur du simulateur');
          writeln ('n''est pas compris entre 1 et ',largmax,'
 (initsimul)');
          readln
          end
     else if vectvide(t)
             then begin
                  writeln ('erreur, le vecteur-définition est
vide (simulateur)');
                  readln
                  end
```

```
     else if not egalg
             then begin
                  writeln ('erreur, le vecteur de noms de variable
ne comprend pas autant');
                  writeln ('de variables que le tableau de résultats
 (simulateur)');
                  readln
                  end
              else if vectlg(t) <> x
                   then begin
                        writeln ('erreur, le tableau de résultats
 contient plus de variables');
                        writeln ('qu''il y a de paramètres passés
au calculateur (simulateur)');
                        readln
                        end
               else if not memevect
                    then begin
                         writeln ('erreur, le vecteur de noms de
variable ne comprend pas les mêmes');
                         writeln   ('noms  de  variable  que  le
 tableau de résultats (simulateur)');
                         readln
                         end
                    else begin
                         okvi (nom,bon);
                         if not bon
                            then begin
                                 writeln ('erreur,  la colonne de
 valeurs correspondant a la variable indépendante');
                                 writeln  ( '"',nom,'"   faisant
partie du schéma expérimental  est vide (simulateur)');
                                 readln
                                 end
                            else begin
                                 elargir (t, vectlg(t) - lrmat(t),bon);
                                 if not bon
                                   then begin
                                        writeln ('erreur, il n''y
 a plus suffisamment de place pour stocker');
                                        writeln ('les résultats de
la simulation (simulateur)');
                                        readln
                                        end
                                    else begin
                              ok := true;
                              i := 1; bon := true;
                              while   bon and  (i <= largmax) do
                                if chaine[i] <> ''
                                 then begin
                                      j := 1;
                                      iemedef (t,def,j);
                                      while chaine[i] <> def.nom do
                                        begin
                                        j := j + 1;
                                        iemedef(t,def,j);
```

```
                                    end;
                                  vectncol[i] := j;
                                  i := i + 1;
                                  end
                               else bon := false;
                        end
  end
  end
end;



procedure calculvinc(var tab:tabres; i,j : integer);

  begin

  end;



procedure lirepi(chaine:vectstring; var t:tabres; l:integer);

  var val : real;
      i, nocoli : integer;
      def: variable;
      bon : boolean;


   procedure majve (var tab : tabres; ncol, lg : integer);

     var val : real;
         def: variable;

     begin
        iemedef (t,def,ncol);
        if lg = 1 then   val := def.origine
                  else  lire (tab, ncol, lg-1, val);
        ecrire (tab, ncol, lg, val);
     end;


begin
  i := 1;  bon := true;
  while bon and (i <= largmax) do
   if chaine[i] <> ''
     then begin
          nocoli := vectncol[i];
          iemedef (t,def,nocoli);
          if def.genre = ve
            then majve (t,nocoli,1);
          if (def.genre = vi) and (def.md = nonc)
            then  calculvinc (t, nocoli,1);
          lire (t,nocoli,1,val);
          p[i] := val;
```

```
          i := i + 1;
          end
     else bon := false;
  end;



procedure ecrirepi (chaine:vectstring; var t:tabres;l:integer);

  var i, nocoli : integer;
      def : variable;
      bon : boolean;

  begin
   i := 1;  bon := true;
   while (i <= largmax) and bon do
    if chaine[i] <> ''
       then begin
            nocoli := vectncol[i];
            iemedef (t,def,nocoli);
            if (def.genre = vd) or (def.genre = ve)
              then ecrire (t,nocoli,l,p[i]);
            i := i + 1;
            end
       else bon := false;
  end;



procedure liberemat (var tab : tabres);

  var court, preced : lien;
      suivt, prec, cour, long, debut : integer;

 begin
  if not exist (tab)
     then begin
            writeln ('erreur, la matrice est vide (liberemat)');
            readln
          end
     else begin
  long := tab.lg;
  debut := tab.pt^.data;
  cour := tabgen.libre;
  if debut < cour
   then if debut + long = cour
          then
               if cour = longmax + 1
                 then begin
                      tabgen.libre := debut;
                      tabgen.tabval[debut,1] := long;
                      tabgen.tabval[debut,2] := 0
                      end
                 else begin
```

```
                tabgen.tabval[debut,1] := long +
                                          tabgen.tabval[cour,1];
                tabgen.tabval[debut,2] := tabgen.tabval[cour,2];
                tabgen.tabval[cour,1] := 0;
                tabgen.tabval[cour,2] := 0;
                tabgen.libre := debut
              end
          else begin
                 tabgen.tabval[debut,2] := cour;
                 tabgen.tabval[debut,1] := long;
                 tabgen.libre := debut
               end
else begin
       while cour < debut do
         begin
           prec := cour;
           cour := trunc (tabgen.tabval[cour,2]);
           suivt := trunc ( tabgen.tabval[cour,2])
         end;
       if (tabgen.tabval[prec,1] + prec = debut) and
           (debut + long = cour)
            then begin
                   tabgen.tabval[prec,1] := tabgen.tabval[prec,1]
                                    + long + tabgen.tabval[cour,1];
                   tabgen.tabval[prec,2] := suivt;
                   tabgen.tabval[debut,1] := 0;
                   tabgen.tabval[debut,2] := 0;
                   tabgen.tabval[cour,1] := 0;
                   tabgen.tabval[cour,2] := 0;
                 end
            else if tabgen.tabval[prec,1] + prec = debut
                   then begin
                          tabgen.tabval[prec,1] :=
                          tabgen.tabval[prec,1] + long;
                          tabgen.tabval[prec,2] := cour;
                          tabgen.tabval[debut,1] := 0;
                          tabgen.tabval[debut,2] := 0
                        end
                   else if debut + long =  cour
                          then begin
                                 tabgen.tabval[debut,1] := long +
                                 tabgen.tabval[cour,1];
                                 tabgen.tabval[debut,2] :=
                                 tabgen.tabval[cour,2];
                                 tabgen.tabval[prec,2] := debut;
                                 tabgen.tabval[cour,1] := 0;
                                 tabgen.tabval[cour,2] := 0
                               end
                          else begin
                                 tabgen.tabval[prec,2] := debut;
                                 tabgen.tabval[debut,1] := long;
                                 tabgen.tabval[debut,2] := cour
                               end;

         end;
new(court);
```

```
    new(preced);
    if tab.pt = tabgen.premier
                then tabgen.premier := tabgen.premier^.suivt
                                else begin
                                       court := tabgen.premier;
                                       while court <> tab.pt do
                                          begin
                                            preced := court;
                                            court := court^.suivt;
                                          end;
                                        preced^.suivt := court^.suivt
                                       end;

    end;
    end;



procedure creermat (var tab : tabres; nbrelignes : integer;
                    nbrecol : integer;  var ok : boolean);


    var debut : integer;
        nouv, preced, pt : lien;



    procedure faireplace;

      var x, y, z, w :integer;
          compt, nfree, suivt, cour : integer;
          pt : lien;


      begin
       if tabgen.libre <= longmax then begin
       compt := 0;
       cour := tabgen.libre;
       suivt := trunc(tabgen.tabval[cour,2]);
       nfree := trunc (tabgen.tabval[cour,1]);
       pt := tabgen.premier;
       x := cour + nfree - 1;
      while pt^.data < x  do
       pt := pt^.suivt;
      while suivt <> 0 do
         begin
           nfree := trunc (tabgen.tabval[cour,1]);
           compt := compt + nfree;
           y := cour + nfree;
           z := cour + nfree - compt;
           w := (suivt - (cour + nfree))*(6*largmax);
           move (tabgen.tabval[y,1], tabgen.tabval[z,1], w);
           x := cour + nfree - 1;
           while ((pt^.data )= x) and (pt^.data  < suivt)) do
```

```
            begin
              pt^.data := pt^.data - compt;
              pt := pt^.suivt;
            end;
          tabgen.libre := tabgen.libre + (suivt - (cour + nfree));
          cour := suivt;
          suivt := trunc(tabgen.tabval[cour,2]);
        end;

    for x := (tabgen.libre) to longmax do
        begin
         for y := 1 to largmax do
         tabgen.tabval[x,y] := 0;
        end;

  tabgen.tabval[tabgen.libre,1] := longmax - tabgen.libre + 1;
   end;
end;



procedure trouveplace (n:integer; var noligne : integer;
                       var trouve : boolean );

 var preced, cour, suivt , x, nfree : integer;



 begin
 trouve := false;
 if tabgen.libre = longmax + 1
           then
           else begin
 cour := tabgen.libre;
 suivt := trunc (tabgen.tabval[cour,2]);
 nfree := trunc (tabgen.tabval[cour,1]);
 while (suivt <> 0) and ( trouve = false) do
    begin
      if n <= nfree
           then begin
                 trouve := true;
                 noligne := cour;
                 if nfree = n
                    then if cour = tabgen.libre
                              then tabgen.libre := suivt
                              else tabgen.tabval[preced,2] := suivt
                    else  begin
                            if cour = tabgen.libre
                               then tabgen.libre := cour + n;
                            tabgen.tabval[cour + n, 1] := nfree - n;
                            tabgen.tabval[cour + n, 2] := suivt;
                          end
                 end
           else begin
```

```
                    preced := cour;
                    cour := suivt;
                    suivt := trunc (tabgen.tabval[cour,2]);
                    nfree := trunc (tabgen.tabval[cour,1]);
                   end;
      end;
  if trouve = false
      then begin
            if n > (longmax - cour + 1) then begin
            faireplace;
            if  n <= (longmax - tabgen.libre + 1)
               then begin
                     noligne := tabgen.libre;
                     tabgen.libre := tabgen.libre + n;
                     x := tabgen.libre;
                     if tabgen.libre <> longmax + 1 then begin
                     tabgen.tabval[x,1] := longmax - x + 1;
                     tabgen.tabval[x,2] := 0;
                                                    end;
                     trouve := true;
                    end;
            end
                                        else begin
            trouve := true;
            noligne := cour;
            if n = (longmax - cour + 1 )
              then if tabgen.libre = cour
                      then tabgen.libre := longmax + 1
                      else tabgen.tabval[preced,2] := 0
              else begin
                     tabgen.tabval[cour + n,1] := longmax
                                              - (cour +n) + 1;
                     tabgen.tabval[cour + n, 2] := 0;
                     if tabgen.libre = cour
                        then tabgen.libre := tabgen.libre + n
                        else tabgen.tabval[preced,2] := cour + n;
                    end
            end;
            end;

  end;
  end;



  begin
    ok := false;
    if  nbrelignes <= 0 then write ('erreur,  le nombre de lignes
donné est incohérent')
                        else if nbrelignes > longmax
                                then write ('erreur,  il n''y  a
plus assez de place pour créér la matrice')
                                else begin
    if  nbrecol  >  largmax then write  ('erreur,  le  nombre  de
 colonnes demandé est supérieur à la limite permise')
                          else if nbrecol <= 0
```

```
                                        then write ('erreur,  le nombre
de colonnes donné est incohérent')
                                        else begin
    trouveplace (nbrelignes, debut, ok);
    if ok then begin
    tab.lg := nbrelignes;
    tab.lr := nbrecol;
    new (pt);
    new (preced);
    new (nouv);
    nouv^.data := debut;
    pt := tabgen.premier;
    if pt^.data > debut
       then begin
             nouv^.suivt := tabgen.premier;
             nouv^.tab := addr(tab);
             tabgen.premier := nouv
            end
        else begin
              while pt^.data < debut do
                 begin
                   preced := pt;
                   pt := pt^.suivt
                 end;
               nouv^.suivt := preced^.suivt;
               nouv^.tab := addr(tab);
               preced^.suivt := nouv
              end;
    tab.pt := nouv;
    end
    else write ('erreur, il n''y a plus assez de place pour créér
la matrice');
    end;
  end;
 end;



function assezplace ( var tab:tabres; lig,col: integer): boolean;

   var bon : boolean;


   begin
     if exist(tab)
       then begin
         writeln ('erreur, la matrice n''est pas vide (assezplace)');
         readln
         end
       else begin
     bon:= true;
     if (col > largmax) or (col <=0) or (lig<=0) or (lig > longmax)
      then assezplace:= false
      else begin
           creermat(tab,lig,col,bon);
```

```
          if bon = false
               then assezplace := false
               else begin
                    assezplace := true;
                    liberemat(tab)
                    end
          end
          end
  end;



procedure creerpas ( var tab : tabres; b1, b2, pas : real);

  var  lig, col: integer;
       deb, fin, cour : real;
        bon : boolean;

  begin
   bon := false;
   if exist (tab) then begin
                       writeln ('erreur, la future matrice n''est
 pas vide (creerpas)');
                       readln
                       end
                  else begin
   if b1 <= b2 then begin
                    deb := b1;
                    fin := b2;
                    end
               else begin
                    deb := b2;
                    fin := b1;
                    end;
   lig :=0;
   cour := deb;
   while cour <= fin do
     begin
      lig := lig + 1;
      cour := cour + abs(pas)
     end;
   col := 1;
   creermat (tab, lig, col, bon);
   if bon = true then  begin
     lig := 0;
     col := 1;
     cour := deb;
     while cour <= fin do
       begin
         lig := lig + 1;
         ecrire ( tab, col, lig, cour);
         cour := cour + abs(pas);
       end;
       end
```

```
      else begin
           writeln (' (creerpas)');
           readln
           end;
   end;
end;



procedure juxtaval( var tab1, tab2, tab3: tabres);

  var bon :boolean;
      lig,col: integer;
      val: real;

 begin

    for lig := 1 to lgmat(tab1) do
       for col := 1 to lrmat(tab1) do
          begin
            lire (tab1, col, lig, val);
            ecrire (tab3, col, lig, val)
          end;
     for lig := 1 to lgmat(tab2) do
        for col := 1 to lrmat(tab2) do
          begin
            lire (tab2, col, lig, val);
            ecrire (tab3, col + lrmat(tab1), lig, val)
          end;

   end;



procedure juxtamat (var tab1, tab2, tab3 : tabres);


 var bon : boolean;

 begin
  if not exist(tab1)
     then begin
            writeln ('erreur, la première matrice originale
est vide (juxtamat)');
            readln
          end
     else if not exist(tab2)
             then begin
                    writeln ('erreur, la seconde matrice originale
est vide (juxtamat)');
                    readln
```

```
                    end
                else  if exist (tab3)
                          then begin
                                writeln ('erreur, la future matrice
n''est pas vide (juxtamat)');
                                readln
                                end
                          else begin

   bon := false;
   if lgmat(tab1) <> lgmat(tab2)
    then begin
         writeln ('erreur, les deux matrices originales n''ont pas');
         writeln ('le même nombre de lignes ! (juxtamat)');
         readln
        end
    else  begin
          creermat (tab3, lgmat(tab1), lrmat(tab1) + lrmat(tab2),bon);
          if bon = false then  begin
                               writeln (' (juxtamat)');
                               readln
                               end
                          else  juxtaval(tab1,tab2,tab3);
          end;
   end;
   end;



procedure juxtadef( var tab1,tab2,tab3:tabres);

  var i:integer;
      vari: variable;


  begin
   if not vectvide (tab3) then initvect (tab3);
   for i:=1 to vectlg(tab1) do
     begin
     iemedef(tab1,vari,i);
     ecriredef(tab3,vari,i);
     end;
   for i:= 1 to vectlg(tab2) do
     begin
     iemedef(tab2,vari,i);
     ecriredef(tab3,vari,i);
     end;
  end;
```

```
procedure juxtavect  (var tab1, tab2, tab3: tabres);


   begin
     if vectvide(tab1)
       then begin
            writeln ('erreur, le vecteur-définition du premier
tableau original est vide (juxtavect)');
            readln
            end
       else if vectvide(tab2)
        then begin
             writeln ('erreur, le vecteur-définition du second
tableau original est vide (juxtavect)');
             readln
             end
        else if not veriflgvect (tab3, vectlg(tab1) + vectlg(tab2))
               then begin
                    writeln ('erreur, le futur vecteur-définition
 serait trop long (juxtavect)');
                    readln
                    end
               else if mememom(tab1,tab2)
                     then begin
                          writeln ('erreur, le futur vecteur-
définition  contiendrait deux variables');
                          writeln ('de même nom (juxtavect)');
                          readln
                          end
                     else juxtadef(tab1,tab2,tab3);
    end;



procedure juxtatab (var tab1,tab2,tab3 : tabres);

  var bon : boolean;


  begin
   if not coherence(tab1)
     then begin
        writeln ('erreur,  le premier tableau original n''est pas
 cohérent (juxtatab)');
        readln
        end
      else if not coherence(tab2)
         then begin
              writeln ('erreur, le second tableau original n''est
pas cohérent (juxtatab)');
              readln
              end
         else  if exist (tab3)
```

```
                    then begin
                         writeln ('erreur, la future matrice n''est
 pas vide (juxtatab)');
                         readln
                         end
                     else begin

    bon := false;
    if lgmat(tab1) <> lgmat(tab2)
     then begin  writeln ('erreur, les matrices originales n''ont pas')
;
                writeln ('le même nombre de lignes ! (juxtatab)');
                readln
       end
     else if not veriflgvect (tab3, vectlg(tab1) + vectlg(tab2))
            then begin
                 writeln ('erreur, le futur vecteur-définition
serait trop long (juxtatab)');
                 readln
                 end
            else if memenom(tab1,tab2)
                 then begin
                         writeln ('erreur, le futur vecteur-définition
 contiendrait deux variables ');
                         writeln ('de même nom (juxtatab)');
                         readln
                         end
                     else begin
                         creermat (tab3, lgmat(tab1),
                                   lrmat(tab1) + lrmat(tab2),bon);
                         if not bon
                           then begin
                                writeln (' (juxtatab)');
                                readln
                                end
                           else begin
                                juxtaval (tab1, tab2,tab3);
                                juxtadef (tab1,tab2,tab3)
                                end
                         end;
    end;
   end;



procedure copierval (var tab1,tab2 : tabres);

  var lig, col:integer;
      val: real;

  begin
    for lig := 1 to lgmat(tab1) do
      for col := 1 to lrmat(tab1) do
        begin
        lire(tab1,col,lig,val);
```

```
        ecrire(tab2,col,lig,val)
        end
  end;


procedure copiermat (var tab1,tab2 : tabres);

  var bon : boolean;

  begin
    bon := true;
    if not exist(tab1)
      then  begin
       writeln ('erreur, la matrice originale est vide (copiermat)');
       readln
            end
      else if  exist(tab2)
         then begin
              writeln ('erreur, la future matrice n''est pas vide
(copiermat)');
              readln
              end
         else begin
              creermat(tab2, lgmat(tab1),lrmat(tab1),bon);
              if not bon then begin
                              writeln (' (copiermat)');
                              readln
                              end
                         else copierval (tab1,tab2);
              end

    end;


procedure copierdef (var tab1,tab2:tabres);

  var i:integer;
      def:variable;

  begin
    if not vectvide(tab2) then initvect(tab2);
    for i := 1 to vectlg(tab1) do
      begin
      iemedef(tab1,def,i);
      ecriredef(tab2,def,i)
      end
  end;
```

```
procedure copiervect (var tab1,tab2:tabres);


  begin
   if vectvide (tab1)
      then  begin
            writeln  ('erreur, le vecteur-definition original
est vide (copiervect)');
            readln
            end
      else if not veriflgvect (tab2,vectlg(tab1))
             then begin
                  writeln ('erreur, le futur vecteur-definition
serait trop long (copiervect)');
                  readln
                  end
             else copierdef (tab1,tab2)
  end;



procedure copiertab (var tab1,tab2 : tabres);

  var bon:boolean;

  begin
   bon := true;
   if not exist(tab1)
      then begin
      writeln ('erreur, la  matrice originale est vide (copiertab)');
      readln
           end
      else if exist(tab2)
         then begin
              writeln ('erreur, la future matrice n''est pas
vide (copiertab)');
              readln
              end
          else if vectvide (tab1)
                then begin
                     writeln  ('erreur, le vecteur-definition
original est vide (copiertab)');
                     readln
                     end
                else if not veriflgvect (tab2,vectlg(tab1))
                       then begin
                            writeln ('erreur, le futur vecteur-
definition serait trop long (copiertab)');
                            readln
                            end
                       else begin
                         creermat(tab2, lgmat(tab1),lrmat(tab1),bon);
                         if not bon then begin
                                         writeln (' (copiertab)');
```

```
                                                    readln
                                                    end
                                               else begin
                                                    copierval (tab1,tab2);
                                                    copierdef (tab1,tab2)
                                                    end
                                    end
  end;



procedure prodval (var tab1,tab2,tab3 : tabres);

  var lig1,lig2,col1,col2,i : integer;
      val : real;


  begin
    for lig1 := 1 to lgmat(tab1) do
      for lig2 := 1 to lgmat(tab2) do
        begin
        for col1 := 1 to lrmat(tab1) do
         begin
         lire (tab1, col1, lig1, val);
         ecrire (tab3, col1, (lgmat(tab2) * (lig1 - 1)) + lig2, val)
         end;
        for col2 := 1 to lrmat(tab2) do
         begin
         lire(tab2, col2, lig2, val);
         ecrire(tab3,lrmat(tab1)+col2,((lig1-1)*lgmat(tab2))+lig2,val)
         end;

        end
end;



procedure prodmat (var tab1,tab2,tab3 :tabres);

  var bon : boolean;

  begin
      bon := true;
      if not exist(tab1)
       then begin
              writeln ('erreur, la première matrice originale
est vide (prodmat)');
              readln
              end
       else if not exist(tab2)
              then begin
                     writeln ('erreur, la seconde matrice originale
```

```
 est vide (prodmat)');
                        readln
                       end
                 else  if exist (tab3)
                        then begin
                             writeln ('erreur, la future matrice
n''est pas vide (prodmat)');
                             readln
                             end
                       else begin
                          creermat(tab3, lgmat(tab1) * lgmat(tab2),
                                   lrmat(tab1) + lrmat(tab2),bon);
                          if not bon then begin
                                          writeln (' (prodmat)');
                                          readln
                                          end
                                     else prodval (tab1,tab2,tab3)
                             end
    end;



  procedure prodtab (var tab1,tab2,tab3: tabres);

   var bon : boolean;

   begin
     bon := true;
     if not coherence(tab1)
      then begin
        writeln ('erreur,  le premier tableau original n''est pas
cohérent (prodtab)');
        readln
        end
      else if not coherence(tab2)
         then begin
              writeln  ('le  second tableau original  n''est  pas
cohérent (prodtab)');
              readln
              end
         else  if exist (tab3)
                 then begin
                      writeln ('erreur, la future matrice n''est
 pas vide (prodtab)');
                      readln
                      end
     else if not veriflgvect (tab3, vectlg(tab1) + vectlg(tab2))
            then begin
                 writeln ('erreur, le futur vecteur-définition
 serait trop long (prodtab)');
                 readln
                 end
            else if mememom(tab1,tab2)
                  then begin
```

```
                        writeln ('erreur, le futur vecteur-définition
contiendrait deux variables');
                        writeln ('de même nom (prodtab)');
                        readln
                        end
                    else begin
                        creermat (tab3, lgmat(tab1)*lgmat(tab2),
                                  lrmat(tab1) + lrmat(tab2),bon);
                        if not bon
                          then begin
                               writeln (' (prodtab)');
                               readln
                               end
                          else begin
                               prodval(tab1,tab2,tab3);
                               juxtadef (tab1,tab2,tab3)
                               end
                        end
end;
```

ANNEXE 5 : TEXTE DU DIDACTICIEL

```
procedure biochem (var at,bt,ct,dt: real; k1,k2 ,td: real);

  var f1,f2: real;
      i:integer;

 begin
   f1 := k1 * at * bt * td;
   f2 := k2 * ct * dt * td;
   at := at - f1 + f2;
   bt := bt - f1 + f2;
   ct := ct + f1 - f2;
   dt := dt + f1 - f2 ;
   if (at < 0) or (bt < 0) or (ct < 0) or (dt < 0)
   then begin
        at := 0;
        bt := 0;
        ct := 0;
        dt := 0
        end;
 end;



var k1lu,k2lu, dtcal : real;



procedure calcul1 (var at, bt, ct, dt: real; T : real);

  begin
    biochem (at,bt,ct,dt,k1lu, k2lu, dtcal);
  end;



procedure simulat1 (var t: tabres; chaine: vectstring);

  var l : integer;
      ok: boolean;

begin
  initsimul(chaine,t,ok,5);
  if ok
    then for l := 1 to lgmat(t) do
      begin
      lirepi (chaine,t,l);
      calcul1 (p[1], p[2], p[3], p[4], p[5]);
      ecrirepi (chaine,t,l)
      end
end;
```

```
program reactbioch;

{$I c:systeme.pas}
{$I c:biochimie.pas}

var tab: tabres;
    def:variable;
    str: stringmax;
    tfin, tpas, a0, b0, c0, d0: real;
    encore, ok : boolean;
    chaine: vectstring;
    nb : integer;



  procedure lirek (var k: real; x: integer);

   var ok : boolean;

   begin
   ok := false;
   while not ok do
      begin
      lirenombre (k,20,x);
      if (k <= 0) or (k )1)
         then begin
              gotoxy (1,20);
              write ('ce nombre n''est pas acceptable, recommencez')
              end
         else ok := true;
      end
     end;



procedure lireconc (var conc:real; x:integer);

 var ok:boolean;


begin
ok:= false;
while not ok do
     begin
     lirenombre (conc,46, x);
     if (conc <=0) or (conc )= 100)
       then begin
            gotoxy(1,20);
            write ('ce nombre n''est pas acceptable, recommencez')
            end
       else ok := true;
     end
 end;
```

```
procedure liretemps (var temps:real);

 var ok: boolean;

 begin
   ok := false;
   while not ok do
     begin
     lirenombre (temps,39,7);
     if temps <= 0
       then begin
            gotoxy(1,20);
            write ('ce nombre n''est pas acceptable, recommencez')
            end
       else ok := true;
     end
   end;



procedure lireint (var int:real);

 var ok: boolean;

 begin
   ok := false;
   while not ok do
     begin
     lirenombre (int,39,8);
     if (int <= 0)  or (int > 0.1)
       then begin
            gotoxy(1,20);
            write ('ce nombre n''est pas acceptable, recommencez')
            end
       else ok := true;
     end
   end;



procedure calculfois (fin, pas:real; var nomb:integer);

  var i : integer;
      cour : real;

  begin
    i := 0;
    cour := 0;
    while cour <= fin do
        begin
        i := i + 1;
        cour := cour + pas
```

```
        end;
    nomb := i
end;



begin

  initialisation;
  ok := false;
  repeat
    inittab (tab);
    initchaine (chaine);
    clrscr;
    writeln ('K1 ( )0 et ( 1 ) :                    / sec.M');
    writeln ('K2 ( )0 et ( 1 ) :                    / sec.M');
    writeln ('concentration initiale de A ( )=0 et (100) :
     M');
    writeln ('concentration initiale de B ( )=0 et (100) :
     M');
    writeln ('concentration initiale de C ( )=0 et (100) :
     M');
    writeln ('concentration initiale de D ( )=0 et (100) :
     M');
    writeln ('temps total de l''expérience  ( )0 ) :
     seconde(s)');
    writeln ('intervalle de temps ( )0 et (=0.1 ) :
     seconde');
    lirek (k1lu, 1);
    lirek(k2lu,2);
    lireconc (a0,3);
    lireconc (b0,4);
    lireconc (c0,5);
    lireconc (d0,6);
    liretemps (tfin);
    lireint (tpas);
    clrscr;
    calculfois (tfin,tpas,nb);
    if not assezplace (tab, nb, 1)
       then writeln ('l''expérience serait trop longue; elle est
    annulée')
       else
          begin
          creerpas(tab,0,tfin,tpas);
          dtcal := tpas;
          deficon (def,'temps','seconde');
          ajoutdef (tab,def);
          defevol(def,'[A]','M',a0);
          ajoutdef(tab,def);
          defevol(def,'[B]','M',b0);
          ajoutdef(tab,def);
          defevol(def,'[C]','M',c0);
          ajoutdef(tab,def);
          defevol(def,'[D]','M',d0);
          ajoutdef(tab,def);
```

```
        ajoutnom(chaine,'[A]');
        ajoutnom(chaine,'[B]');
        ajoutnom(chaine,'[C]');
        ajoutnom(chaine,'[D]');
        ajoutnom (chaine,'temps');
        simulat1 (tab,chaine);
        writeln ('pour voyager dans la matrice, utilisez les touches
        représentant');
        writeln ('les flèches; pour en sortir, tapez <return>');
        gotoxy (1,24);
        writeln ('tapez <return> pour continuer');
        readln;
        clrscr;
        affichemat(tab);
        liberemat(tab);
        clrscr
        end;
   gotoxy (1,2);
   writeln ('voulez-vous recommencer (o/n)? :');
   lirestring (str, 33,2,1,['o','n']);
           if str= 'o' then encore := true
                       else encore := false;
 until not encore;
end.
```

BIBLIOGRAPHIE

[SPAIN] : SPAIN J. D., 1982, BASIC Microcomputer Models in Biology, Addison-Wesley Publishing Company, London.

[RANDALL] : RANDALL J. E., 1980, Microcomputers and Physiological Simulation, Addison-Wesley Publishing Company, London.

[LEBRET] : LEBRETON J. D., MILLIER C., 1982, modèles dynamiques déterministes en biologie, Masson, Paris.